Universität Rostock

Traditio et Innovatio

Generische ITS-Softwarearchitektur und ITS-Prozess

Dissertation
zur Erlangung des Grades
Doktor-Ingenieur (Dr.-Ing.)
am Institut für Informatik
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Nikolaj Troels Graf von Malotky
Rostock, 2020

https://doi.org/10.18453/rosdok_id00002843

Dissertation

Fakultät für Informatik und Elektrotechnik

Angaben über die Dissertation

Titel: Generische ITS-Softwarearchitektur und ITS-Prozess

Autor: Nikolaj Troels Graf von Malotky

Fakultät: Fakultät für Informatik und Elektrotechnik

Gutachter:

Prof. Dr.-Ing. Alke Martens
Universität Rostock
Institut für Informatik

Prof. Dr.-Ing. habil. Peter Forbrig
Universität Rostock
Institut für Informatik

Prof. Dr. habil. Andreas Harrer
Fachhochschule Dortmund
Fachbereich Informatik

Jahr der Einreichung: 2020

Jahr der Verteidigung: 2020

Lebenslauf des Autors

Akademische und schulische Laufbahn

2000 - 2003	Georg-von-Giesche-Oberschule in Berlin Abschluss: Mittlere Reife
2003 - 2006	Sophie-Charlotte-Oberschule (Gymnasium) in Berlin Abschluss: Allgemeine Hochschulreife
2006 - 2012	Bachelorstudium Informatik an der FU-Berlin Nebenfach: Physik Abschluss: Bachelor of Computer Science (Informatik)
2012 - 2014	Masterstudium Informatik an der FU-Berlin Nebenfach: Mathematik Vertiefungsgebiet: Medizininformatik Abschluss: Master of Computer Science (Informatik)

Berufslaufbahn

2010 - 2014	Unzählige Studentenjobs
2014 - 2015	Wissenschaftliche Hilfskraft an der Universität Rostock am Lehrstuhl der praktischen Informatik
2015 - 2017	Wissenschaftlicher Mitarbeiter an der Universität Rostock am Lehrstuhl der praktischen Informatik in Forschung und Entwicklung einer hyperspektralen Wundgewebekamera in Zusammenarbeit mit der Firma Diaspective Vision
2017 - 2018	Wissenschaftlicher Mitarbeiter an der Universität Rostock am Lehrstuhl der praktischen Informatik in Forschung und Entwicklung eines hyperspektralen Sportsensors in Zusammenarbeit mit der Firma OXY4
2018 - 2020	Wissenschaftlicher Mitarbeiter an der Universität Rostock im Juniorstudium für die Weiterentwicklung des Systems und die Verwaltung (Beschaffung, Personal und Organisation)
ab 2019	Wissenschaftlicher Mitarbeiter an der Universität Rostock im Projekt DigiCare für die Entwicklung eines intelligenten Lehr-/Lernsystems in der Pflege

Abstrakt

In dieser Arbeit wird ein neuer Ansatz für eine ITS-Softwarearchitektur entwickelt. Anhand einer ausgiebigen Analyse vorhandener Softwarearchitekturen werden wiederkehrenden Probleme dieser aufgezeigt. Ein ITS ist durch seine KI-gesteuerte Adaptivität an einem Lernenden das effektivste Lehrsystem. Es integriert idealtypisch den Interaktionsbereich zwischen Lehrenden und Lernenden ab - den Lehrprozess. Zudem ist es auch prädestiniert dafür, als ein Werkzeug der Didaktikforschung verwendet zu werden um vergleichbare empirische Daten zu erheben. Als Softwaresystem besteht es aus einer seit den 1980er Jahren immer wieder verwendeten Anzahl an Systembausteinen, die im Kern gleichgeblieben sind, in ihrer Funktionalität und Komposition aber quasi evolutionär weiterentwickelt worden sind. Bis heute gibt es keine wiederverwendbare Architektur intelligenter Lehr-/Lernsysteme.

Lerneffektforschung und Systemforschung benötigen beide einen Standard, um die Divergenz zukünftiger Systeme zu minimieren. Zu diesem Zweck wird an dem Vorbild eines menschlichen Lehrers der Lehrprozess abgeleitet und für Intelligent Tutoring Systems (ITS) angepasst. Darauf aufbauend wird eine neue Softwarearchitektur für ITS entwickelt. Diese beiden Dinge beschreiben den allgemein gültigen Aufbau und Funktionsweise eines ITS und erleichtern das Verständnis solch eines Systems auch für Nicht-Softwareentwickler. Aufbauend auf der Softwarearchitektur und dem Prozess wird ein Softwareframework entwickelt, welches eine Umsetzung eines ITS erleichtert. Ein Prototyp stellt exemplarisch eine Implementierung der Realisierbarkeit unter Beweis.

Abstract

In this thesis a new approach for an ITS software architecture is developed. Based on an extensive analysis of existing software architectures, recurring problems of these architectures are identified. An ITS is the most effective teaching system due to its AI-controlled adaptivity on a learner. It ideally integrates the area of interaction between teacher and learner - the teaching process. It is also predestined to be used as a tool for didactics research in order to collect comparable empirical data. As a software system, it consists of a number of system components that have been used repeatedly since the 1980s. Although the core of the system has remained the same, its functionality and composition have been further developed in a quasi evolutionary way. Until today there is no reusable architecture of intelligent teaching/learning systems.

Learning effects research and systems research both require a standard to minimize the divergence of future systems. For this purpose, the teaching process is derived from the model of a human teacher and adapted for Intelligent Tutoring Systems (ITS). Based on this, a new software architecture for ITS will be developed. These two things describe the generally valid structure and functionality of an ITS and facilitate the understanding of such a system also for non-software developers. Based on the software architecture and the process, a software framework is developed that facilitates the implementation of an ITS. A prototype is used as an example to demonstrate the feasibility of implementing the framework.

Schlagwörter

- Technology enhanced learning (TEL)
- Intelligent tutoring systems (ITS)
- Intelligente Lehr-/Lernsysteme (ILLS)
- E-Learning
- Software architecture
- Software engineering

CR-Klassifikation (1998)

- D.2.13 Reusable Software : Domain engineering
- D.2.11 Software Architectures : Patterns
- K.3.1 Computer Uses in Education

Inhaltsverzeichnis

I. EINFÜHRUNG UND THESEN

I.1. Einführung

I.1.a. Einleitung

Verglichen mit der Zeit, in der komplexes Leben auf der Erde existiert, besteht der Mensch in der Geschichte der Erde erst seit Kurzem und doch hat er die Vorherrschaft über den Planeten erreicht, wie es keine Spezies vor ihm geschafft hat. Außergewöhnlich ist die Anzahl an Neuronen im menschlichen Gehirn von 86 Billionen [2012 HERCULANO-HOUZEL], eine Zahl die kein Tier erreicht. Doch auch der klügste Mensch würde es in der Natur ohne Vorwissen und Werkzeug sehr schwer haben, wenn man bedenkt dass das Gehirn 500 Kilokalorien am Tag benötigt [2012 HERCULANO-HOUZEL] und diese ohne die Verwendung von Werkzeug nur schwer zu beschaffen sind. Selbst ein Werkzeug für Reparaturen muss verstanden werden. Effizientere Werkzeuge sind komplexer und erfordern mehr Menschen mit Wissen die zusammenarbeiten. Die Menschheit hat es durch Wissen geschafft viele Ziele zu erreichen. Jede Generation kann mehr schaffen als die vorherige, weil sie auf das vorhandene Wissen aufbauen kann. Doch solch ein anwendbares Wissen benötigt Zeit, um im einzelnen Individuum abrufbar zu sein.

Unsere Gesellschaft ist geradezu darauf ausgelegt, dass deren Mitglieder einen gewissen Wissensstand erreichen. Deutschland hat 1948 festgelegt in der „Allgemeine Erklärung der Menschenrechte“, Artikel 26, Absatz 1: „Jeder hat das Recht auf Bildung.[...]“. Die Grundschulbildung und teilweise auch die weitere Schul- und Berufsausbildung muss von Privathaushalten nicht direkt bezahlt werden. Wissen hat eine hohe Priorität in Deutschland und auch in vielen anderen Ländern der Welt. Dies sieht man auch in dem Postulat des Lebenslangen Lernens [1984 STRZELEWICZ].

Lehren im Sinne des Vermitteln von Wissen (Fertigkeiten und Fähigkeiten) und Lernen gehörten oft unmittelbar zusammen. Im Tierreich kann man beobachten wie durch Zeigen und Nachahmung gelehrt wird. Ähnliches gibt es auch beim Menschen, zum Beispiel erforscht in den Konzepten des „Learning by Doing“ [2017 KNOLL] oder „Cognitive Apprenticeship“ [1977 BANDURA]. Diese Ansätze haben sich in vielen Bereichen bewährt. Allerdings ist beispielsweise reines Lernen durch Beobachtung fehleranfällig, da ein Lernender nicht genau weiß auf was er achten muss und nur lernen kann, was auch passiert. Ein wichtiges Hilfsmittel für effizienteres Lehren ist die höhere Kommunikation und mit ihr auch die Sprache. Mit Sprache ist es möglich Lehrinhalte zu vermitteln, welche abstrakter sind. Damit ist das Wissen noch immer an ein Individuum gebunden. Mit der Entwicklung von Zeichnungen und Schrift, war es möglich Wissen zu persistieren und transferieren. Lehrmaterial bleibt ohne einen Lehrer erhalten und es kann auch ohne diesen gelernt werden. Doch dieses Lehrmaterial ist statisch, verbraucht viel Platz und ist nur in den seltensten Fällen interaktiv. Automaten können dieses Problem teilweise lösen, aber erst mit dem Computer ist es möglich ein relativ kostengünstiges System zu erschaffen, dass auf kleinstem Raum, sowie zeit- und ortsunabhängig komplexen Lehrstoff interaktiv und angepasst an den jeweiligen Lerner vermitteln kann. Dieses System soll keineswegs ein Mittel zum Ersetzen eines menschlichen Lehrers sein. Ziel ist es dem Lernenden oder Gruppen von Lernenden eine weitere Option zu geben, Lehrmaterial ohne Lehrer zu verstehen, auch wenn es nur zur Übung, Vertiefung, für die Phasen zwischen persönlichem Unterricht oder übergangsweise für Gebiete mit zu wenig Lehrern ist. Jede Verbesserung in der Bildung, bei dem ein Computersystem helfen kann, ist wünschenswert. Denn ein aufgeklärter Mensch ist der Schüssel zur Entwicklung des Potenzials des Einzelnen und der Menschheit im Gesamten.

I.1.b. Motivation

Als erstes stellt sich die Frage: Warum überhaupt einen Computer nutzen, anstatt direkt persönlich und mit Kreide, Tafel, Stift und Papier zu unterrichten? Dafür gibt es viele Gründe, die sicherlich kontrovers diskutiert werden könnten. Einige Gründe sind folgend aufgezählt.

- Lehrern ein anderes und neues Lehrmittel zu geben, welches für bestimmte Bereiche besser geeignet ist als die bisherigen
- Interesse am Lernen der Lernenden durch den Technikeinsatz erhöhen
- Ermöglichung von Selbststudium
- Kosten verringern
- Lehrmaterialien digital bereitstellen und verwalten
- Überall und jederzeit verfügbare Lehre (Zeit- und Ortsunabhängigkeit) und damit neue Zielgruppen (zum Beispiel Berufstätige) zu erreichen [2004 BREMER]
- Stabile nachvollziehbare Lehreinsätze
- Einfache Sammlung der Forschungsdaten
- Als Aushilfe für einfache Betreuung von Tutorien
- Einen höheren Lernerfolg sicherzustellen [2004 BREMER] [1991 SHUTE]
- Motivation bei den Lernenden zu erhöhen
- Kooperationsmöglichkeiten mit anderen Hochschulen, Fachbereichen und Ländern auszuschöpfen [2004 BREMER]

Beispielsweise im Kontext der universitären Bildung gibt es zwei Sichtweisen. Die eine Sichtweise betont die Wichtigkeit des Campuslebens, die Kooperation „in real life“ mit Kommilitonen und den Kontakt (Kommunikation) mit den Dozenten. Die andere Sichtweise stellt dar, dass beispielsweise in Großstädten die Anfahrtszeiten teilweise enorm sind, die Hörsäle vollständig überfüllt und im Semester zu viel Zeit für Verwaltungsthemen für die Zeit und Örtlichkeit verloren gehen. Weiterhin bieten schlecht oder wenig didaktisch ausgebildete Dozenten anspruchslose Vorträge von bekanntem oder fehlerhaftem Material dar. Die Geschwindigkeit von Lehrveranstaltungen sind angepasst auf den Durchschnitt einer teils über 180 Personengruppe aus unterschiedlichen Fachbereichen. Dabei sind selten bis keine Unterbrechungen oder Fragen möglich. Hinzu kommt schlechte Akustik und aus der persönlichen Sicht die Scham nachzufragen. Studierende erleben im Zeitalter der Digitalisierung, dass viele dieser Defizite mit einem Computer behebbar sind. Auch wenn ein Computer als Lernutensil seine eigenen Nachteile mit sich bringt, so gibt es sicherlich einen Einsatzzweck in der autonomen Lehre für einen gewissen Teil der Lernenden. So kann man heute feststellen, dass akribisch und genau geschaut werden muss, wann Studierenden der Einsatz digitaler Lehrmaterialien mehr hilft, als die Anwesenheit auf einem Campus. Diese Diskussion ist empfindlich und wird in den öffentlichen und in den wissenschaftlichen Medien kontrovers diskutiert - an dieser Stelle kann diese Diskussion nicht weiter geführt werden. Nachfolgend aufgeführte Aspekte sind jedoch aus der Perspektive dieser Arbeit wichtig.

Einer der Beweggründe für das Interesse an computergestütztem Lernen ist ein Lehrkräftemangel, welcher in den nächsten Jahren voraussichtlich schlimmer werden wird [2016 UNESCO]. Besonders Fachbereiche, welche auf Regeln basieren, könnten durch eine Unterstützung der Lehre durch ein Lehr-/Lernsystem entlastet werden. Vor allen in den Fächern, in denen besonders Lehrer gesucht werden, wie Mathematik, Physik, Informatik und dergleichen. Dabei soll nicht die einzelne Lehrkraft ersetzt werden, sondern die Lernenden zum Selbststudium motiviert und gefördert werden [2002 KERRES]. Der Lernende, wie auch der Lehrer wären damit zeitlich flexibler und in dieser Phase des Lernens auch nicht örtlich aneinander gebunden. Es wäre eine hybride Lehrstruktur möglich [2004 BREMER]. Zudem ergeben sich die Vorteile durch das computerunterstützte Lernen, darunter das multimediale Lernen, den geringeren psychischen Druck durch einen

virtuellen Akteur und das Experimentieren an einem möglichst realitätsnahen Modell, anstatt Versuchsaufbauten in der Realität. Der letzte Punkt ist besonders relevant in gefährlichen oder sehr wichtigen Vorgängen, in denen die Gesundheit von anderen Menschen beteiligt sind oder kostspieliges Gerät verwendet wird. In einigen Bereichen wird solch ein System schon eingesetzt, besonders in Bereichen, bei denen die Handlungsfreiheit des Lernenden eingeschränkt ist. Es ist an der Forschung zu zeigen, dass sich diese Systeme für den Lerner und die Lehrkräfte lohnen und dies auch bei komplexeren Aufgaben möglich ist.

Einige Verhaltensweisen von menschlichen Lehrern sind dadurch bedingt, dass diese selten 1:1 unterrichten können. Wie in Abbildung R-I1b-1 zu sehen wird im 1:1-Unterricht effektiver gelernt als im klassischem Frontal-1:30-Unterricht. Da es aufgrund von zu hohen Kosten und Lehrkräftemangel unmöglich wäre für jeden einzelnen Lerner einen Domänenexperten bereit zu stellen, wären Computer, welche die Lehrtätigkeit wenigstens teilweise übernehmen können, ein gutes Lehrinstrument. Ein intelligentes Lehrsystem übertrumpft damit ein reines Klassenszenario und kann die Leistung von Anfängertutoren erreichen [2012 RAI and BECK].

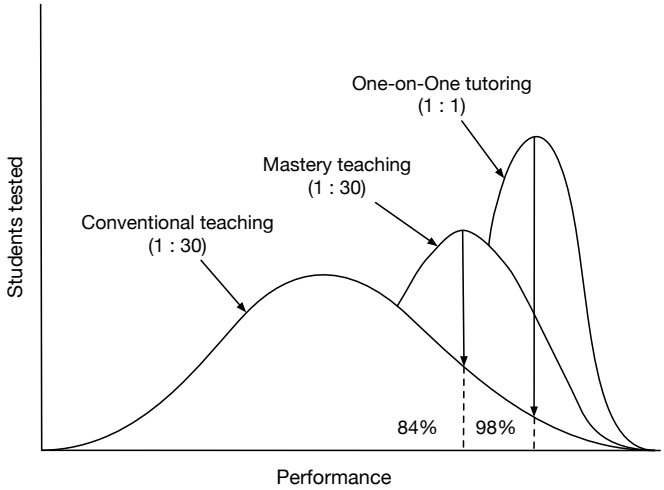


Abbildung R-I1b-1: Vorteile von 1:1 Unterricht [2010 WOOLF]

Wenn man 1:1 mit 1:30 im Unterricht vergleicht, dann verändert man nicht nur die Anzahl sondern meistens auch die Lehrstruktur. Frontalunterricht mit einer Person versus Frontalunterricht mit 30 Personen wäre vergleichbar. Der Lehrer verändert sein Verhalten und passt es an die gegebenen Umstände an. Dies verändert aber das wissenschaftliche Setting und verschlechtert die Vergleichbarkeit. Menschliche Lehrer sind auch weniger adaptiv bei Lernenden mit wenig Vorwissen [2010 CHI and ROY]. Ein intelligentes System könnte dahingehend optimiert werden und bevorzugt keine besseren Lernenden mit besserer Lehre.

In einer Studie [2009 MENDICINO, RAZZAQ and HEFFERNAN] sind die Tests nach einer Hausarbeit von den Teilnehmern besser beantwortet worden, welche anstatt mit Stift und Papier mit einem intelligenten Lehr-/Lernsystem gearbeitet haben. In einer anderen Arbeit war der Wissenszuwachs mit einem ITS auch höher als ohne [2010 BEAL, ARROYO and COHEN]. Ein Grund dafür könnte die direkte Antwort und Hilfe auf ihre Lösungen sein, welche durch ein ITS gegeben werden. Diese Aussage wird auch durch eine andere Studie gestützt, bei dem Lehrer proportional mehr Zeit für die Hausaufgabenkontrolle aufgewandt haben und die Schüler in einem nachträglichen Test geringere Werte erzielt haben, als Schüler welche mit dem ITS gelernt haben [2011 SINGH, SALEEM and PRADHAN]. Eine weitere Studie hat die Effizienz von ITS sogar fast so hoch wie die eines Lehrers eingeordnet [2011 VAN LEHN] und das mit einer Aussage die aus Studien kombiniert wurde.

Die Möglichkeit, Teile der Lehre einem Computersystem zu überlassen klingt dramatisch, doch soll klargestellt werden, dass nicht Lehrer ersetzt werden. Das Computersystem sollte als ergänzendes Lehrmittel eingesetzt werden, dass dann zum Einsatz kommt, wenn einer der folgenden Punkte zutrifft.

- Es einem Lehrer aufgrund der Situation nicht möglich ist die Aufmerksamkeit für den einzelnen Lernenden bereit zu stellen.
- Lernenden nicht in der Lage sind, direkt mit einem Lehrenden zu interagieren (Gründe könnten die Veranstaltungsform, der Aufenthaltsort des Lehrenden oder Lernende, aber auch Krankheit oder pflegende Verpflichtungen sein)
- Lehrmaterialien in Ergänzung zu Präsenzveranstaltungen bereitstehen
- Experimente oder Information vermittelt werden sollen, die nicht in Realexperimenten oder direkten Erfahrungen vermittelbar sind (z.B. Ausbreitung von Waldbrand durch Simulation)

Eine große Interessengruppe bei den Lernenden sind die Personen, welche auf ein Selbststudium angewiesen sind. Darunter fallen Berufstätige, Pflegebedürftige, entfernt lebende Personen oder (Allein-)erziehende. Der Vorteil von einem E-Learningsystem liegt in seiner Flexibilität mit Blick auf den Ort, die Zeit, die Dauer der Bereitstellung des Materials und den Inhalt. Es ist möglich mitten in der Nacht im Zug 4 Stunden lang aktiven Unterricht zu einem speziellen Thema zu bekommen. Damit lässt sich die Lernzeit und Intensität selbstständig einteilen, der Lernort teilweise (in Abhängigkeit vom Endgerät) selbst wählen. Diese Flexibilisierung erklärt in erster Linie den Hype um sämtliche E-Learning Formate in den 1990er Jahren. Die Grenzen der Systeme sind dann ein Hinweis darauf, warum dieser Hype nur knappe zehn Jahre gehalten hat. Das Wiederaufleben der digitalen Lehr- und Lernsysteme seit wenigen Jahren ist ein Zeichen für die technische Weiterentwicklung und auch für die auf den Hype folgende Ernüchterung, die zu der Umsetzung gangbarer und praktikabler Wege geführt hat. Seit wenigen Jahren gehörten E-Learning Systeme der vielfältigsten Art dem Alltag an - nach wie vor im Spannungsfeld zwischen Nutzbarkeit, Effektivität und teilweise massiv aufwändiger Entwicklung.

Am effizientesten aus lernpsychologischer Sicht sind interaktive Lehrmaterialien (Quelle Diss Martens). Im Gegensatz zu einfachen „page turnern“, die eher eine Form digitalisierten Materials darstellen, fordern interaktive Lehrmaterialien den Lernenden auf, mit dem Material aktiv zu arbeiten. Der Unterschied von einem normalen interaktivem Lehrmaterial zu einem intelligentem Lern- und Lehrsystem ist der Grad der Adaptivität und Interaktivität. Die Erwartungen an ein Lehr-/Lernsystem sind durch ihre komplexe Natur hoch. Um diese auch effizient erfüllen zu können, benötigt man einen abstrakten Plan von dem was man softwaretechnisch entwickeln möchte. Ein grober Plan über den Aufbau einer solchen Software würde den Einstieg und die Entwicklung sowohl in der Entwicklungszeit, -qualität als auch in den Kosten positiv beeinflussen.

Gründe für den Einsatz von einem Computer als Lernwerkzeug sind neben dem Lehrkräftemangel die Forschung am Lehren und Lernen. Um Aussagen über das Verhalten und die Lerngeschwindigkeit zu machen, müssen die Testbedingungen möglichst immer gleich sein, bis auf den zu untersuchenden Faktor. Ein Computer kann weitaus besser vergleichbare Situationen erschaffen als ein menschlicher Lehrer, weil die Ausführung sowohl genauestens festgelegt, als auch protokolliert werden kann. Dies haben Psychologen — vor allem Kognitionsforscher — erkannt und sich zunutze gemacht [2001 ANDERSON]. Das Beobachterphänomen kann bei einem computerbasierten Experiment wegfallen. Eine Protokollierung von allen Interaktionen mit der virtuellen Lehrkraft ist für die spätere Auswertung in jedem Fall erforderlich und kostet damit kaum zusätzlichen Aufwand. Das Verhalten des Computers ist bei gewollten Wiederholungen immer gleich, im Gegensatz zu dem potenziell ungeduldigen menschlichen Lehrer. Zudem lassen sich Verhaltensweisen eines virtuellen Lehrsystems leicht austauschen um Unterschiede oder Korrelationen zu erforschen. Für die Forschung sind solche Systeme neben dem praktischen

Nutzen in der Anwendung deshalb auch besonders außerhalb der Informatik relevant, da sie in mehreren Fachbereichen (zum Beispiel Lernpsychologie, Inklusionsforschung, Didaktikforschung) untersuchen, wie ein Lehrer modelliert werden kann und wie ein Mensch effizient lehrt und lernt. Innerhalb der Informatik liegt mit einem computergesteuerten Lernsystem ein interaktives System vor, bei dem Bereiche wie Software Engineering, Effizienz von Verarbeitungen, Mensch-Maschine-Interaktion, Simulationssysteme und Datenhaltung bzw. -auswertung erforscht werden können. Ein intelligentes Lern-/Lehrsystem ist damit für die Forschung als Forschungswerkzeug in der Didaktik und Lernpsychologie relevant und in der Informatik als Gegenstand der Forschung. Neben Forschung und Lehre gibt es weiteres Interessengruppen für gut durchdachte, sauber entwickelte digitale Lehr-/Lernsysteme. Das Konzept zur Entwicklung solch eines Systems ist für die Firmen interessant, welche solch ein System anpassen oder neu implementieren möchten. Das resultierte System als Produkt ist in Lehreinrichtungen einzusetzen und hat wiederum andere Bedarfe zu erfüllen sowie andere Vor- und Nachteile. Es gibt also vieles, was für die Entwicklung und Weiterentwicklung von digitalen Lehr-/Lernsystemen spricht.

Es mag verwunderlich sein, dass bei dem massiven Bedarf an digitalen Lehr-/Lernsystemen die hochpotenten intelligenten Lehr-/Lernsysteme noch nicht massentauglich sind und so gut wie nie kommerziell entwickelt werden. Bei einem zukünftigen Boom von jener Art von Systemen, von welchem man durch die immer mehr in den Alltag integrierten Assistenten - basierend auf künstlicher Intelligenz - ausgehen kann (zum Beispiel Google Assistant [2020 GOOGLE], Alexa [2020 AMAZON], Siri [2020 APPLE INCORPORATED], Cortana [2020 MICROSOFT CORPORATION]) wäre ein gemeinsamer Nenner für intelligente Lehr-/Lernsysteme von großem Vorteil. Eine Entwicklung eines Pre-Standards, der unter Umständen den Grundstein für weitere Entwicklungen bildet oder dazu beiträgt, erspart Arbeit und erzeugt eine Vergleichbarkeit der Systeme und der von ihnen generierten Daten. Dies wurde erkannt, hat aber nur zur Entwicklung von Standardisierungsvorhaben geführt [2004 MARTENS], die für die tatsächliche Entwicklung von Software wenig zielführend sind. Denn die Entwicklung der künstlichen Intelligenz, welche mit Menschen interagiert, wie zum Beispiel die interaktiven Sprachassistenten Siri oder Alexa, hängt häufig von bereits vorhandenen großen Datenmengen ab. Ein Standard erlaubt es in verschiedenen Situationen und Domänen das Wissen in der Datenbank unabhängig von der Präsentation systematisch zu strukturieren und gemeinsam zu sammeln. Nachträglich ist es dann möglich über verschiedene Schnittstellen die Struktur zu konvertieren, bereit zu stellen und die Vergleichbarkeit in Spezialfällen für andere Vorteile einzutauschen. Umgekehrt wären der Aufwand und der Verlust der Zuverlässigkeit durch die Konvertierung womöglich zu hoch.

Das Ziel ist also eine Vereinheitlichung zu erschaffen, bevor der Markt von Adhoc-Entwicklungen überschwemmt wird. Dieser Zeitpunkt ist jetzt gekommen. Der Autors dieser Doktorarbeit hat jahrelang an Expertensystemen in der Medizin gearbeitet [2017a GRAF VON MALOTKY, NAGORSNICK and MARTENS][2017b GRAF VON MALOTKY, NAGORSNICK and MARTENS][2017 GRAF VON MALOTKY, NAGORSNICK and NICOLAY][2017a MARTENS, NICOLAY and GRAF VON MALOTKY][2017b MARTENS, NICOLAY and GRAF VON MALOTKY][2019 WASSMANN, GRAF VON MALOTKY and MARTENS] und entwickelt in seinem aktuellen Projekt „DigiCare“ ein ITS mit [2020b GRAF VON MALOTKY and MARTENS]. Der Erfinder der ersten ITS-Softwarearchitektur hatte den gleichen Forschungshintergrund von medizinischen Expertensystemen, dessen Komponenten bis heute Bestand haben. Aus diesen Gründen bildet das Vorwissen des Autors der Doktorarbeit eine gute Basis für die Entwicklung einer Softwarearchitektur.

I.1.c. Erkanntes Problem

Im Rahmen der Entwicklung dieser Dissertation wurden zwei zentrale Probleme lokalisiert und bearbeitet:

1. Es gibt keine ITS Architektur im Sinne eines sauberen Software Engineering.
2. Es gibt keine durchdachte Abbildung des Lehr-/Lernprozesses in ITS, der wiederverwendbar und unabhängig von der Anwendungsdomäne ist.

Um ITS als Softwaretyp im Bereich des digitalen Lehrens- und Lernens dauerhaft ausserhalb experimenteller lernpsychologischer Forschung zu etablieren, benötigt es beide Teilaspekte. Die genannten Probleme werden im Folgenden weiter ausgeführt.

Bei der Erschaffung eines neuen digitalen Lehr-/Lernsystems werden häufig die gleichen Fragen gestellt. Genau wie die Entwurfsmuster, welche für die Programmierung wiederkehrender algorithmischer Probleme Lösungsmuster anbietet, so gibt es für neue Systeme entworfene Softwarearchitekturen für bekannte Arbeitsbereiche eines Systems. Viele ITS haben keine formal festgelegte Architektur für ihre Zwecke verwendet, auch wenn sie grundlegend die gleiche Basisarchitektur oder deren Komponenten verwenden. Teilweise ist das darin begründbar, dass sich die ITSs sehr stark voneinander unterscheiden können. Bei einer Neuentwicklung wird ein weiteres ITS erschaffen, welches sich grundlegend nur sehr wenig von den vorherigen Architekturen unterscheidet, aber nicht ähnlich genug zu vorhanden ITS ist um deren Architektur zu übernehmen. Hinzu kommt, dass alle bestehenden Architekturbeschreibungen vage und formal nicht immer korrekt sind (siehe auch [2004 MARTENS]). Anstatt ständig ähnliche Architekturen zu entwickeln, wäre es besser, wenn es eine Architektur gibt, welche nicht zu speziell ist, jedoch alle relevanten Grundlagenkonzepte integriert. Dabei muss gleichzeitig Raum für die Fortentwicklung bleiben. Gebraucht wird ein Modell eines intelligenten Lehr-/Lernsystems, dass flexibel genug ist, um den nötigen Anpassungen gerecht zu werden, als auch ein Grundgerüst liefert, welches bei einem Systementwurf hilft.

Intelligente Lehr-/Lernsysteme sind bisher eher selten als ein Lehrmittel gewählt worden, weil der Entwicklungsaufwand für Lehrinhalte zu aufwändig ist. Einen Schritt weiter gedacht soll das in dieser Dissertation entwickelte Modell nicht nur aufgezeigt werden, um Softwarearchitekturen zu entwickeln, sondern es soll auch direkt in eine Anwendung implementierter sein. Es müssen daher auch Vorteile in der Implementierung vorhanden sein, um den Aufwand und Kosten gegenüber einer Neuentwicklung ohne vorherige Architektur zu verringern. Denn die Entwicklung eines ITS erfordert beträchtliche Personenstunden sowohl von den Autoren des Domänenwissens als auch von Programmierern. Es wurde geschätzt, dass jede Lehrstunde eines ITS ungefähr 200 bis 300 Stunden Entwicklung benötigen [1999 MURRAY][2003 MURRAY]. Werkzeuge die speziell für Domänenautoren und Entwickler geschaffen werden, können diesen Aufwand auf 50-100 Stunden reduzieren [2009 ALEVEN, MCLAREN and SEWALL]. 50-100 Stunden ist für einige Domänen kosteneffizient, da diese Lehrstunde nicht statisch ist und an genügend vielen Lernenden wiederholt werden kann. Diese Kosteneinsparung kann mit einem zusätzlich bereit gestelltem Softwareframework erreicht werden, in dem die Grundkomponenten schon vorher implementiert sind. Sodass es bei dem Erstellen von einem neuen ITS die Grundlagen und der Einstieg erleichtert wird ohne bekannte Prinzipien zu vernachlässigen.

I.1.d. Problemspezifikation

Formulierung des Problems

Es existieren nur vage generische Softwarearchitekturen oder sehr spezifische (siehe Kapitel „Probleme der gefundenen ITS-Softwarearchitekturen“). Die Grundstruktur wurde vor mehr als 20 Jahren definiert und hat sich bisher kaum verändert, die klassische ITS-Softwarearchitektur. Es fehlt eine genauer definierte, allgemein anwendbare Softwarearchitektur. Neuere Softwarearchitekturen beziehen sich meist auf eine domänenspezifische Implementation eines ITS. Es gibt in den letzten 10 Jahren nur ein begrenztes Bemühen um eine Weiterentwicklung der generalisierten Architektur.

Wissenschaftliche Relevanz

Für die wissenschaftliche Auswertung von verschiedenen ITSs und damit auch von deren Lehrmethoden ist eine Vergleichbarkeit unerlässlich. Bei Implementierung von verschiedenen ITSs mit gleicher Softwarearchitektur oder sogar dem gleichen Softwareframework würde die Vergleichbarkeit erheblich gesteigert werden. Dies wäre sowohl für die Lehrforschung relevant, als auch für die Architekturbeschreibung von ITSs im Allgemeinen. Systeme mit dieser Architektur lassen sich durch denselben Aufbau vergleichen, was wiederum auch die Auswertungen der Ergebnisse von ITS auf Lernverhaltensweisen verbessert. All diese Einzelpunkte können untersucht und bewertet werden, um damit der Weiterentwicklung von ITS beizutragen. Diese Punkte zusammen in einer Arbeit unter einem Thema zu bearbeiten sorgt in Summe für eine höhere wissenschaftliche Relevanz.

Wissenschaftlicher Beitrag

Es werden in dieser Arbeit die Grundlagen für ITS aufgearbeitet und auch eine Referenz für die benutzten Fachbegriffe erstellt um den Einstieg in das komplexe Thema zu vereinfachen. Diese Arbeit vergleicht zusätzlich auch die Softwarearchitektur von ITS und deren Geschichte. Zuerst entsteht ein generisches Konzept mit einer Spezifikation für ITS. Das erklärte Ziel ist, aus dem Konzept dann eine Softwarearchitektur für ITS zu entwickeln, welches breit anwendbar und trotzdem genauer ist, als das bisherige klassische Modell.- Für die Entwicklung eines ITS-Prototypen als Umsetzung der Softwarearchitektur wird erst eine wiederverwendbares Softwareframework entwickelt, um damit eine Implementierung grundlegender Funktionen zu liefern. Darauf aufbauend wird der Prototyp entwickeln. Es wird für ITS eine grobe Struktur, den Stil, die groben Elemente, die Kommunikation der groben Elemente untereinander und wie sie wieder in größere Elemente eingeordnet werden und mit anderen größeren Teilen kommunizieren und wie das ganze in die Lehre passt aufgezeigt. Der wissenschaftliche Beitrag ist damit im Bereich des Softwareengineering angesiedelt und zeigt eine generische Lösung in verschiedenen Schichten auf.

Womit sich diese Arbeit nicht befasst

Diese Arbeit beschäftigt sich nicht mit der idealen Umsetzung nach didaktischen und psychologischen Maßstäben. Es wird nicht untersucht oder beschrieben was didaktisch sinnvoll ist. Das betrifft beide Enden der Implementierung, sowohl die Benutzeroberfläche als auch die Schnittstelle und konkrete Ausprägung der Software zum Nutzer sowie die Implementierung im Speziellen auf Seiten der Persistierung der Daten, wie zum Beispiel welchen Dateityp und Aufbau die Datenbanken haben oder wie die Benutzeroberfläche generell oder im Einzelfall aussehen soll. Es wird auch keine produktreife Software programmiert, denn der Beitrag dieser Arbeit ist hauptsächlich nicht im Prototypen sondern im Konzept und der Architektur angesiedelt.

I.2. Thesen

Das Forschungsziel ist es einen Beitrag in der formalen Standarisierung von wichtigen Konzepten im ITS-Forschungsbereich zu leisten. Auch wenn ein höherer Lehrerfolg im Vergleich zu anderen E-Learningsystemen ein positives, angestrebtes Ergebnis ist, so ist unabhängig davon die verbesserte Vergleichbarkeit zwischen ITS eines der Kernziele, um ITSs als ein Werkzeug für den Vergleich von pädagogischen und didaktischen Regeln zu nutzen.

Um Ziele zu setzen, die überprüft werden können, wird das Problem in einzelne Thesen transformiert. Die unten folgenden Thesen werden von dieser Arbeit aufgestellt und mittels der Analyse und Konzeption auf ihren Wahrheitsgehalt überprüft. Die Thesen 1 bis 3 sind zusammenfassend über einen allgemein wiederverwendbaren ITS-Lehrprozess, während die Thesen 4 bis 6 die allgemein wiederverwendbare Struktur von ITS durch eine Architektur behandelt. Die Zusammenführung der beiden Bereiche kommt in These 7 zum tragen. Die Implementierung kommt ist Fokus der letzten beiden Thesen, wobei der Einsatz der Architektur in einem Softwareframework in These 8 vorkommt, während der Einsatz aller Thesen in einem Beispiel-ITS in These 9 zur Geltung kommt.

Für die Weiterentwicklung eines allgemeinen Standards ist die Überprüfung der Thesen unerlässlich. Sind sie wahr so ist es möglich an der Umsetzung einer generellen wiederverwendbaren Standards für Prozess und Architektur zu arbeiten die sowohl im abstrakten als auch im konkreten praktischen Fall zu arbeiten. Bei genügend möglicher Detailtiefe kommt nicht nur der ITS-Entwicklung, sondern auch der Forschung zu Gute. Ein echter Standard hätte für ITSs viele Vorteile und müssten dann von einem Konsortium wie IEEE oder DIN beschlossen werden um breit gefächert genutzt zu werden. Einen Teil der Vorarbeit dafür kann allerdings schon jetzt gemacht werden, wie ein formal verifizierbares Modell, dass Rückschlüsse absichert. Mit einem Standard sind der Wortschatz und die Definition für alle die ihm folgen gleich und wenn das darin enthaltene Modell in mehreren Abstraktionsebenen durch die Repräsentation des Systems ohne zu tiefe technische Details beschrieben wird, hat dies sowohl einen positiven Einfluss auf die Kommunikation zwischen Softwareentwicklern und Experten, als auch auf deren Verständnis des Systems. Ein Standard erhöht die Vergleichbarkeit von ITSs untereinander und trägt damit auch zur Vergleichbarkeit der resultierenden ITS-Forschung bei. Um die Vorarbeit in einem Pre-Standard auf seine Umsetzbarkeit zu überprüfen, kann ein Softwareframework mit dem integrierten Modell verwendet werden. Software welche mit dem Softwareframework geschaffen wird, hat die gleiche Struktur, Definitionen und Abläufe bei gleichzeitiger Verringerung der Modellierungszeit als auch die Implementationsdauer eines neuen ITS. ITSs mit vergleichbaren Ergebnissen erlauben Analysen der Lehrforschung weit über die ITS-Forschung hinaus die die Lehre insgesamt verbessern kann. ITSs sind ein perfektes Werkzeug zur Generierung von Datenmengen für die Lehrforschung, weil sie eine massenhafte, unermüdliche, einheitliche Wiederholbarkeit von Lehrsituationen herstellen können. Das heißt durch Softwareframeworks kostengünstiger erstellte ITSs könnten anhand der gleichen zugrunde liegenden Konzepte ihre Daten anhand nur gewollter Unterschiede (zum Beispiel in der Lehrstrategie) auswerten. ITSs mit einheitlicher Basis würden damit die Bereiche der ITS-Forschung und die Lehrforschung vorantreiben.

These 1

Es lässt sich ein einheitlicher Prozess der Interaktion eines ITS mit einem Lernenden entwickeln.

These 2

Der Prozess aus These 1 kann in verschiedene, unabhängige Detailstufen aufgeteilt werden.

These 3

Der Prozess aus These 1 lässt sich formal spezifizieren.

These 4

Es lässt sich eine einheitliche Architektur für intelligente Lehr-/Lernsysteme entwickeln.

These 5

Die Architektur aus These 4 kann in verschiedene, unabhängige Detailstufen aufgeteilt werden.

These 6

Die Architektur aus These 4 lässt sich formal spezifizieren.

These 7

Der Prozess aus These 3 und die Architektur aus These 6 sind miteinander kompatibel.

These 8

Die Architektur aus These 6 kann in einem objektorientierten Softwareframework implementiert werden.

These 9

Ein ITS kann mit dem Softwareframeworks aus These 8 implementiert werden. Dieses hält die Architekturvorgaben aus These 6 mittels des Softwareframeworks und gleichzeitig ist dessen Prozess festgelegt auf den in These 3 definierten Prozess.

I.3. Material und Methoden

I.3.a. Programme und Programmiersprache

Es wurden verschiedene Werkzeuge verwendet um sowohl diese Dissertation als auch die Software als Resultat jener zu erzeugen. Darunter fällt auch die Programmiersprache als Umsetzung für das Softwareframework.

Texteditor

Die Dissertationsschrift wurde mit „Pages“ von „Apple“ erstellt. Die verwendete Version ist ab Version 5.0.

Diagrammwerkzeug

Für das Erstellen der Diagramme und Grafiken, wurde das Applikation „OmniGraffle“ der Firma „The Omni Group“ verwendet. Die Version dieser Software war zum Zeitpunkt der Benutzung 6.2.5. Grafiken von Quellen wurden für eine bessere Qualität mit diesem Werkzeug nachgebildet, Grafiken ohne Quelle wurden zum Zweck der Arbeit selbst kreiert.

Betriebssystem

Das verwendete Betriebssystem ist „OS X“ab 10.10.0 „Yosemite“(ab Version 10.12 „Sierra“ wird es „macOS“ genannt).

Programmiersprache

Die verwendete Programmiersprache in der das Softwareframework geschrieben wurde, ist die relativ neue Sprache „Swift“ ab Version 1.0.

Entwicklungsumgebung (IDE)

Die verwendete IDE ist „Xcode“ ab der Version 6.5.

I.3.b. Ressourcen

Es wurde auf verschiedene Ressourcen zugegriffen die möglichst eine breite Abdeckung der Forschungen in Feld der ITS haben. Darunter sind besonders die folgenden Referenzen.

Journals

Die „Internationale Journal of Artificial Intelligence in Education“ ist als gute Quelle für Papiere verwendet worden. Die Ergebnisse werden auch als Buchform publiziert und sind im Springer Verlag erhältlich, welches für Wissenschaftler der Universität Rostock kostenfrei ist. Andere interessante Journals bei denen ITS-Forschung im Fokus sind und in denen auch nützliche Papiere gesucht wurden, sind „Journal of Human-Computer studies“ von Academic Press und „User Modeling and User-Adapted Interaction“ von der Universität Essen.

Konferenzen

Die Konferenz für den Forschungsbereich ist die „Intelligent Tutoring System"-Konferenz und stellt den Hauptanlaufpunkt für Papiere über ITSs. Sie wird alle zwei Jahre gehalten und die Ergebnisse welche dort vorgestellt werden, werden jeweils in einem Buch von Springer veröffentlicht. Auch hier ist der Zugriff für Wissenschaftler der Universität Rostock kostenfrei und es konnten alle Ausgaben nach sinnvollen Papieren für diese Arbeit durchsucht werden. In den Jahren dazwischen hat sich eine andere Konferenz etabliert,

die „International Conference on Artificial Intelligence in Education“ (AIED), welche auch untersucht wurde.

Bücher

Einer der Vorteile von den Modellen in der ITS-Forschung ist es, dass die alten Modelle bis heute Bestand haben. Zur Grundlage der ITSs wurden folgende Bücher verwendet:

- [1970 CARBONELL]
- [1982 SLEEMAN and BROWN]
- [1987 WENGER]

Diese geben ein guten Einblick über die Entstehung von ITS und deren theoretischen Grundlagen. Die künstliche Intelligenz, welche in ITSs verwendet werden verändert sich weitaus schneller und ist für konkrete ITSs von großem Interesse, für die Modellierung einer Architektur ist es aber weniger relevant.

I.3.c. Zusätzliche Papiersuche

Zudem wurden Papiere auch noch einzeln gesucht, um Papiere zu finden die womöglich nicht in den Ressourcen enthalten sind. Hierbei waren vorrangig Papiere mit Bezug zu Softwarearchitekturen relevant.

Gesucht wurde auf folgenden Webseiten:

- ERIC http://eric.ed.gov
- Google Scholar http://scholar.google.com
- ResearchGate https://www.researchgate.net
- IEEExplore http://ieeexplore.ieee.org
- SpringerLink link.springer.com

Verwendete Suchbegriffe:

- „eLearning“ und „E-Learning“
- „TEL“ und „Technology Enhanced Learning“
- „GBL“ und „Game Based Learning“
- „AH und „Adaptive Hypermedia“
- „ITS“ und „Intelligent Tutoring System“
- „Smart environment“

Zusätzlich wurden Quellen der gefundenen Papiere mit einbezogen wenn sie in sehr relevanten Teilen verwendet wurden.

I.3.d. Vorgehensweise

Es wird grob skizziert welche Vorgehensweise erdacht wurde um zum Ziel der Doktorarbeit zu kommen. Es ist nicht das Ziel vorhandene ITS-Architekturen für sich zu verbessern, sondern einen gemeinsamen, formalen Nenner zu schaffen. Man soll das größere Ganze sehen, um ITS Forschungsaufgaben erledigen zu lassen die möglichst wenig andere Variablen haben, ohne das Ergebnis zu verzerren. Um sich nicht in einem Bereich in der Tiefe der Materie zu verlieren, wurde absichtlich Stufen und Rahmen festgelegt mit klaren Zielen was gesucht wird. Erst nach Sammlung von genügend Material wird dieses in der Tiefe ausgewertet. Das ermöglicht einen unvoreingenommenen Einblick in dieses Forschungsgebiet mit gleichzeitig immer steigender Komplexität in eine vorbestimmte Richtung.

- Sammlung von Ressourcen zur Entstehung und Idee eines ITS
- Theoretisches Verständnis von ITSs und dessen Grundlagen erlangen

- Vorhandene ITS-Softwarearchitekturen und deren Beschreibung in zeitlicher Reihenfolge sammeln
- Analysieren von den gesammelten ITS-Softwarearchitekturen nach Mustern und Standards in zeitlicher Reihenfolge
- Den Lehrprozess im Allgemeinen analysieren
- Eigenes grundlegendes Konzept für die Umsetzung des Lehrprozesses in einem Softwaresystem entwickeln
- Bestimmung der Eigenschaften/Parameter der gefundenen Problemstellungen/Themen/Archetypen/Gruppen der vorhandenen ITS-Architekturen
- Eine Softwarearchitektur für ein ITS entwickeln welches das Konzept des Lehrprozesses verwendet und dabei die gefundenen Grundprinzipien von ITS nicht vernachlässigt
- Spezifikation der Softwarearchitektur in Prosa
- Detailliertere Darstellung der Softwarearchitektur für ITS
- Verschiedene Ansichten der Unterschiedlichen Beteiligen auf die ITS-Beschreibung ermöglichen
- Formalisierung der Softwarearchitektur
- Umsetzung und konkrete Entscheidungen für das Softwareframework entwickeln
- Softwareframework entwickeln welches den Vorgaben der ITS-Softwarearchitektur gerecht wird
- Erarbeitung eines leichten Themas ohne komplexe künstliche Intelligenz, Datenbanken oder Benutzeroberfläche
- Prototypen mit Hilfe des Softwareframeworks erstellen

Die Doktorarbeit ist so aufgebaut, dass der Leser als roten Faden zusammen mit dem Autor dieser Arbeit sich dem Ziel annähert. Von der Konzeption wichtiger Grundlagen von ITSs über ITS-Architekturen bis hin zum Prototypen. Viele falsche Forschungspfade werden dabei ausgeblendet und nur ein recht geradliniger Weg beschrieben. Die interessantesten Irrwege werden am Ende speziell beschrieben.

II. KONZEPTE

II.1. Lernen und Wissenstransfer

II.1.a. Grundlagendefinitionen

Um überhaupt das eigentliche Thema zu beginnen, müssen zunächst die Grundlagen aufgearbeitet werden. Denn auch diese unterliegen leicht unterschiedlichen Bedeutungen und damit keine Missverständnisse auftreten werden alle wichtigen Begriffe hier vorher analog zu vorhandenen Definitionen [2007 ZINS] definiert. Ein guter Start ist es, mit den Begriffen in der Wissenspyramide anzufangen, welche in Abbildung R-II1a-1 zu sehen sind. Da sich diese Arbeit im Fachbereich der Lehre bewegt, sind die allgemeine Begriffe nur im Lehrkontext relevant. Sowohl die einzelnen Ebenen werden hier definiert als auch die Transformation von einer Ebenen zur einer anderen.

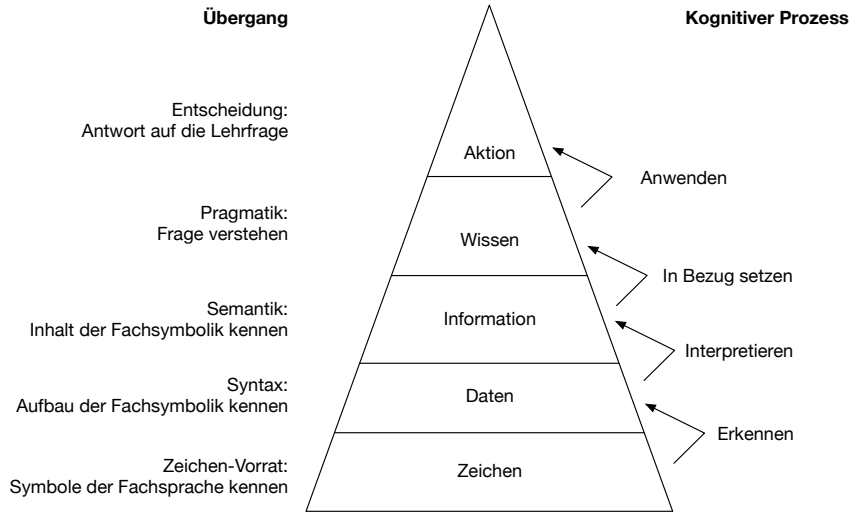


Abbildung R-II1a-1: Wissenspyramide [2001 FUCHS-KITTOWSKI]

Zeichen

Um überhaupt etwas zu verstehen muss das Alphabet bekannt sein, mit dem kommuniziert wird. Hier wird definiert und vereinbart, welche Zeichen oder auch Laute und Signale zulässig sind, in welcher Sprache kommuniziert wird und in welcher Reihenfolge sie gelesen wird. Damit ist nicht nur das ausgesprochene Wort oder Schrift gemeint, sondern jegliche Art Signale und Symbole zu speichern oder zu übertragen. Sei es elektrische Spannungsbereiche in zwei Bereiche zu klassifizieren und die Zeit als Reihenfolge zu nehmen oder chinesische Schriftzeichen von oben nach unten und rechts nach links als Richtung anzuordnen.

Daten

Eine Reihenfolge von Sinneseindrücken die einer Syntax folgt ist ein Datum. Die Syntax enthält nicht nur die erlaubten Zeichen sondern folgt sie auch einer Grammatik. Es gibt Regeln die vorschreiben welche Zeichenfolgen erlaubt sind. Dies ist die letzte Stufe welche noch materialisierbar ist. Daten lassen sich speichern, in Büchern, Festplatten oder anderweitig. Es kann eine Syntax für Bilder, Ton, Haptik oder andere Sinneseindrücke existieren. In der Informatik sind Daten abstrakt nichts weiter als Symbol- oder Zeichenketten.

Information

Die Syntax in den jeweiligen Kontext zu setzen und daraus eine Bedeutung zu erschließen, ist die Semantik und bildet aus Daten eine Information. In der Informatik werden Daten zu Information zusammengesetzt, indem möglichst immer die gleiche Interpretation

von dem Empfänger gewählt wird. Der Empfänger muss dabei aktiv die Daten interpretieren und es ist nicht möglich diese Interpretation in der Syntax zu speichern. Die Syntax kann nicht zu hundertprozentiger Sicherheit korrekt interpretiert werden. Und so kann die Syntax alleine die Semantik nicht enthalten. Eine Information besitzt auch eine Pragmatik, also die Wirkung welche die Information auslöst.

Wissen

In der Allgemeinsprache ist die Definition von Wissen laut Duden die Kenntnis von etwas oder die Gesamtheit der Kenntnisse auf einem bestimmten Gebiet von jemanden [2017 DUDENVERLAG]. Das ist allerdings nicht genau genug. Lebewesen können aus Informationen Wissen generieren, wenn sie miteinander in Bezug gesetzt werden. Meist braucht man eine Vielzahl von wohl gewählten Informationen, welche erst durch ihr Zusammenspiel eine neue Interpretation liefern, die Wissen darstellt. Dazu muss die Bedeutung von den Informationen verstanden worden sein und es gibt verschiedene Möglichkeiten so etwas anzugehen, zum Beispiel kann es hilfreich sein Informationen in Kategorien zu ordnen, sodass dann eine Erkenntnis aus mehreren Informationen gewonnen werden kann. Das Wissen über Informationen kann zu neuen Informationen führen. Deshalb ist es auch logisch, dass der Duden Wissen einer Person zuordnet, weil es eine komplexe - beispielsweise kognitive - Leistung ist, Wissen zu generieren. In der Informatik gibt es Wissen streng genommen nicht, wird aber gerne so verwendet als wäre sie eine Kombination bestimmter Informationen, zum Beispiel in „wissensbasierte Systeme“.

Aktion

Die mehrfache Anwendung des Wissens in unterschiedlichen kontrollierten Umgebungen, mit vielen Möglichkeiten zur Lösung jeder einzelnen, die zu einem als korrekt angesehenem Ergebnis führt stellt ein hinreichend überprüfbaren Test dar, ob Wissen vorhanden ist oder nicht. Es kann zwar anhand der Handlungen nicht eindeutig auf Wissen geschlossen werden, aber bei einer ausreichenden Anzahl an Handlungen, welche für einem vorhandenen Wissensstand erwartet werden, wird trotzdem davon ausgegangen, dass Wissen zumindest in ausreichender Form vorhanden ist. Die richtige Aktion zu wählen um damit ein Problem zu lösen stellt damit die Spitze der Wissenspyramide nach Fuchs-Kittowski [2001 FUCHS-KITTOWSKI] dar.

Domäne

Eine Domäne ist, bezogen auf diese Arbeit, ein Wissensbereich mit eigenem Wissen, in dem Begriffe, Prozesse, Beschreibungen und so weiter enthalten sein können. Es kann von Schulfächern über Fachabteilungen in einer Firma beliebig groß oder klein gefasst werden, solange sich eine Domäne abgrenzen kann, zum Beispiel durch eine Definition der Klassifizierung um zu entscheiden was zu ihr gehört. Eine Domäne kann nur Teil einer anderen sein, oder sich mit einer anderen überschneiden. Als Beispiele sei die Einteilung der Universitätsfächer mit ihren Unterthemen genannt, bei denen es inhaltliche Überschneidungen gibt.

Domänenwissen

Ein Begriff welcher in der ITS-Systemmodellierung auch häufig als „Expertenwissen“ oder „expert knowledge“ bezeichnet wird ist das Domänenwissen. Domänenwissen beinhaltet verschiedene Arten von Wissen über eine Domäne, wie deren Regelsystem und Formeln, Definitionen von Wörtern, Vorschriften, Goldstandards, Standardwerke, Algorithmen, Prozesse oder andere Dinge. Das Domänenwissen spiegelt damit das Wissen eines Fachexperten wider, wobei zu berücksichtigen ist, dass der Fachexperte bei zunehmender Expertise dazu neigt, kognitive Abkürzungen zu wählen. Der Lernende soll idealerweise zum Experten werden, also den Expertenwissensstand be-

herrschen. Es stellt den Teil des Wissens in einem Lehrsystem dar, welches von dem Lernenden, erworben werden soll, nicht aber das Wissen was man braucht um es zu lehren, denn der Experte ist nicht ein Lehrer. Es wird für die Lehre und das Lernen viel mehr Wissen benötigt als im Domänenwissen steckt, deshalb ist es wichtig das Domänenwissen inhaltlich von dem Wissen zu trennen welches nicht direkt das Ziel der Lehre ist. Beispiele für indirekt benötigtes Wissen bei einem Lehrprogramm kann unter anderen sein: Wissen, was man braucht um es zu lehren, wie man mit einem speziellen Lehrsystem umgeht, was die Systemvorraussetzungen sind, wie man ein Lehrsystem design, wie man Lehrsystem programmiert oder wie man Lehrmaterial erschafft.

Definition Solo-Taxonomie

Eine Einteilung für die Stufe des Verständnisses der Lernenden von Domänenwissen ist die SOLO-Taxonomie, siehe Abbildung R-II1a-2. SOLO ist hier ein Akronym für „structure of observed learning outcomes“. Es erlaubt uns eine einfache Kategorisierung in fünf Stufen von dem Wissensstand eines Lernenden oder der Komplexität von Lehrmaterial. Es kann damit auch eingeteilt werden, wie hoch der Komplexitätsgrad von Lehrmaterial ist oder welche Anforderungen an den Lernenden gestellt werden.

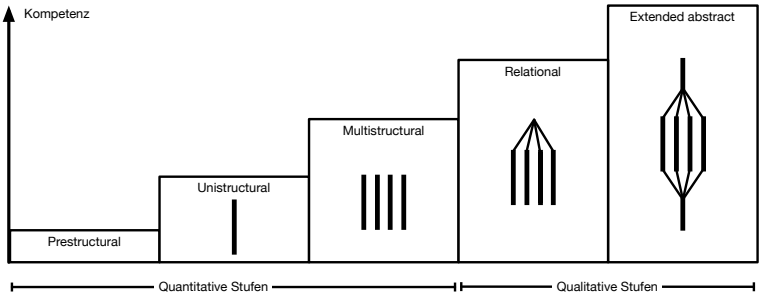


Abbildung R-II1a-2: SOLO-Taxonomie [2014 BIGGS and COLLIS]

Bei der SOLO-Taxonomie wird die Kompetenz beschrieben, welche ein Lernender hat. Mit steigender Stufe, steigt auch die Kompetenz. Die ersten drei Stufen sind quantitativer Natur, während die letzten beiden qualitativ sind. Die erste Stufe nennt sich „Prestructural“. Dabei verstehen die Lernenden zwar die Einzelteile von Wissen, weil sie vorher schon bekannt waren, aber nicht oder nur stark vereinfacht den eigentlichen Inhalt des Wissens. Dies ist zum Beispiel bei rein auswendig gelernten Definitionen, dessen Inhalt nur grob einsortiert werden kann, zu beobachten. Die zweite Stufe nennt sich „Unistructural“. Der Lernende versteht einen Aspekt des Themas für sich genommen, meist einfaches Basiswissen welches relevant ist. Damit ist es dem Lernenden zum Beispiel möglich, das Wissen in anderen Themen wiederzuerkennen und zu identifizieren oder einen einfachen Ablauf zu folgen. Die dritte Stufe heißt „Multistructural“. Der Lernende hat das Verständnis von mehreren Aspekten eines Themas erlangt und kann diese miteinander kombinieren, zum Beispiel sie auflisten und beschreiben oder einem Algorithmus anwenden. Bisher sind diese Aspekte noch getrennt voneinander im Verständnis des Lernenden. Die vierte Stufe heißt „Relational“. Der Lernende kennt jetzt zusätzlich zu den Aspekten vom Thema auch noch deren Zusammenhänge zueinander. Damit hat er das Wissen in einen gemeinsamen Themenkomplex verstanden, der dann auch im gesamten erklärt werden kann. Dies erlaubt ihm die Aspekte zu vergleichen, sie zu unterscheiden, zu sortieren, zu analysieren, Gegensätze zu finden oder Beziehungen zu erkennen (zum Beispiel dass ein Aspekt die Ursache des anderen ist). Die letzte Stufe ist der „Extended Abstract“. Der Lerner kann nun das Wissen in eine höhere Abstraktionsebene transformieren, welches dann auf andere Themenbereiche angewandt werden kann. Das Transformieren in diesem The-

menbereich beinhaltet das Theoretisieren, Verallgemeinern, das Aufstellen von Hypothesen und Reflektieren bekannter Aspekte.

Definition Lernen

Laut Duden [2017 DUDENVERLAG] ist „lernen"definiert als:

- sich Wissen, Kenntnisse aneignen
- sich, seinem Gedächtnis einprägen
- Fertigkeiten erwerben
- im Laufe der Zeit (durch Erfahrungen, Einsichten) zu einer bestimmten Einstellung, einem bestimmten Verhalten gelangen
- (ein Handwerk) erlernen

Ein Lernender (oder auch häufig Schüler oder Lerner genannt, im englischen meistens „student“) ist diejenige Person, welche sich das Wissen, das von einem Experten in der jeweiligen Domäne erstellt worden ist, aneignet. Es ist damit nicht definiert von wem oder was gelernt wird. Damit ist es möglich von einem Computersystem zu lernen. Eine häufig zitierte Ressource [1983 HILGARD] bei der Erklärung was „Lernen“ sei, erklärt Lernen als „Die Veränderung im Verhalten oder im Verhaltenspotential eines Organismus in einer bestimmten Situation, die auf wiederholte Erfahrungen des Organismus in dieser Situation zurückgeht“. In der Lernpsychologie [2008 GERRIG and ZIMBARDO] geht es sogar um eine dauerhafte Verhaltensänderungen, in Abgrenzungen von kurzen, unmittelbaren Verhaltensänderungen, denen kein dauerhafter Lernprozess sonder beispielsweise Ausprobieren zugrunde liegt. Damit fallen viele Verhaltensmuster raus, wie Reflexe, Triebe, Ermüdung, Rausch, angeborene Reaktionen oder Veränderung des Körpers. Zusammengefasst kann man festhalten: Lernen ist ein nicht beobachtbarer Vorgang welcher eine dauerhafte Veränderung des beobachtbaren Verhaltens aufgrund von Erfahrungen darstellt. Die Lerntheorien waren die Ergebnisse der Suche nach Antworten auf die Funktionsweise des Lernens, aus denen dann Lernmodelle entstanden. Schon früh versuchten Psychologen die Kenntnisse über Lernen und bzw. die Zusammenhänge zwischen Lernbedingungen und Lernergebnissen zu systematisieren. Ergebnis dieser Bemühungen waren und sind die Lerntheorien (wie zunächst Behaviorismus und später die kognitionswissenschaftlichen Theorien), aus welchen wiederum die Lernmodelle abgeleitet wurden.

Wissenstransfer

Der Duden definiert „Wissenstransfer“ als Weitergabe von erworbenem Wissen [2017 DUDENVERLAG]. Es ist die Übermittlung des Wissens mittels eines Kommunikationskanals von einem Speicher zu einem anderen. Kommunikationskanäle sind beim Menschen durch seine Sinne festgelegt und der Speicher bei einem Menschen ist sein Gehirn. Die Interaktion mit einem Computerprogramm wie zum Beispiel einem ITS verwendet fast ausschließlich den visuellen und auditiven Sinn. Die Kommunikationskanäle übertragen in diesem Fall Bilder und Text vom Bildschirm und Sprache, Geräusche und Musik durch die Lautsprecher.

Es wird bei dieser Arbeit im speziellen der Wissenstransfer von einem Lehrprogramm zu einem oder mehreren Lernenden betrachtet, unter Umständen auch mit Hilfe eines Lehrers oder Betreuers (auch Supervisor genannt), welche den Wissenstransfer überwacht, aber selbst nicht über das Domänenwissen verfügen muss, siehe dazu Abbildung R-II1a-3. Denn der Wert von Interesse für den Lehrerfolg ist der Wissenszuwachs. Es stellen sich schnell Fragen über Faktoren welche den Wissenszuwachs bei dieser Kombination beeinflussen: „Muss die Interaktion mit dem Programm synchron sein, dass bei einer Antwort das Programm sofort antwortet?“, „Muss das Programm lokal laufen?“, „Sollte der Schüler alleine sein?“ und so weiter. Ein Teil dieser Arbeit widmet sich in den späteren Kapiteln den wichtigsten Faktoren aus der Perspektive des digital gestützten Lehrens und Lernens mit einem adaptiven (intelligente) System.

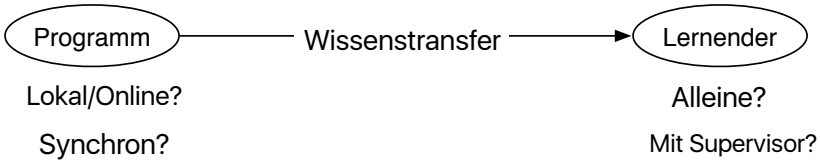


Abbildung R-II1a-3: Wissenstransfer

Der Begriff „Lernen mit Technologien“ (Technology Enhanced Learning TEL) beschreibt bereits im Namen zwei grundlegende Forschungsfelder. An erster Stelle steht dabei die Lernforschung, die untersucht, wie der Prozess des Lernens bei einem Menschen aufgebaut ist (zum Beispiel in den Gebieten Lernpsychologie und Didaktik). An zweiter Stelle steht die Umsetzung als Technik. Technik wird dabei in TEL nicht verstanden als die aus der Didaktik resultierende Methodik, sondern tatsächlich als Einsatz technischer Geräte, im engeren Sinne Computer.

Um Lernen gezielt mit technischen Mitteln zu unterstützen, wird festgelegt welches Wissen von dem Lerner im Speziellen erlangt werden soll. Mit diesem Wissen sollte sich der Lernenden bewusst auseinandersetzen. Dieses Wissen stellt den Corpus des Wissens der Anwendungsdomäne dar und wird Domänenwissen genannt. Damit das technische System dies bewerkstelligen kann, muss das Domänenwissen formalisiert werden und für die Speicherung einer einer Zeichenkette in vorgegebener Syntax in einer Datenbank aufbereitet werden. Erst dann kann ein Autor von Wissen die Datenbank mit Zeichen füllen. Der Autor des Wissens ist damit im Stande, Pragmatik über Semantik und Syntax auf Zeichen in einer Datenbank zu reduzieren (zum Beispiel didaktisch gut aufbereitete Texte zu schreiben und sie per Autorenschnittstelle einer Datenbank hinzuzufügen). Einfache Lehrsysteme dienen dabei als Übertragungsmechanismus, der die Reihenfolge des Inhalts kontrolliert. Ausgeklügeltere Systeme können den Inhalt selbst anpassen.

Der Lernende selbst kennt unter Umständen nicht alle Konventionen und muss es trotzdem schaffen, die ihm gezeigten Zeichen (Text/Audio/Video) in einem kognitiven Prozess wieder in Wissen zu transformieren. Dies geschieht durch die Rekonstruktion des Wissens durch die Ebenen der Pyramide aus Abbildung R-II1a-1 nach oben. Um also Wissen übertragen zu können muss immer der Umweg über die Semantik und dann die Syntax genommen werden, denn nur die Zeichen sind speicherbar. Dies ist quasi die Wissenspyramide, die hier zweimal schrittweise durchlaufen wird: Einmal von oben nach unten (Fachwissen des Lernenden über Formalisierung und Abstraktion zur Computersyntax), dann folgt der Transfer und dann wieder von unten nach oben beim Lernenden (Computersyntax, menschenlesbare Darstellung, Wissensgenerierung im Gehirn), welches durch einen Wissenstransfer geleistet wird, siehe dazu Abbildung R-II1a-4. Überprüfbar für das Lehrsystem ist nur eine Aktion des Lernenden. Die gewählte getätigte Aktion mit dem Lehrsystem wird dann als Lösungsschritt der gestellten Aufgaben gesehen, womit das Wissen des Lernenden geschätzt werden kann.

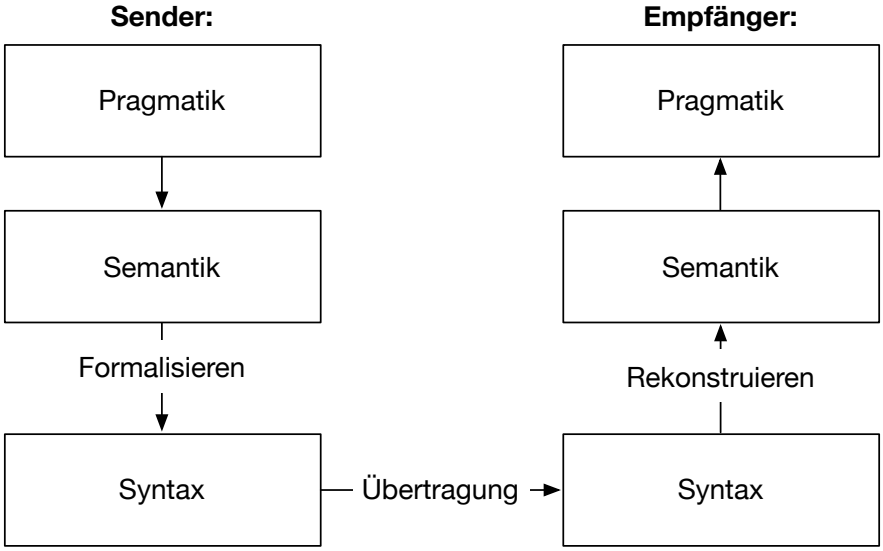


Abbildung R-II1a-4: Übertragung von Wissen [2001 FUCHS-KITTOWSKI]

Formalisiert werden genau diese drei Stufen auf Seiten einer intelligenten Software für die Lehre: die Syntax, die Semantik und die Pragmatik. Syntax ist wie etwas formell aufgeschrieben wird, mit dem Zeichenvorrat und den grammatischen Aufbau der für die Informatik zur Verfügung steht. Semantik ist die Bedeutung oder der Informationsgehalt von einer konkreten Zeichenfolge (zum Beispiel durch Metadaten). Im Falle von einem Lehrprogramm kann dies einfach nur Lehrmaterial in Form von Text sein, welcher einer genau definierten Bedeutung zugeordnet wird, ohne dass der Text selbst vom Lehrprogramm verstanden wird. Die Pragmatik stellt das passende Lehrmaterial bereit, darunter auch Übungen mit Lösungen. Bei einer Übungsaufgabe ist die Beantwortung eine Aktion (ankreuzen, markieren, interagieren und so weiter) für den Lernenden und die Evaluation und Rückmeldung über die Korrektheit der Antwort darauf eine Aktion des Lehrsystems seinerseits. Um die zum Lernen nötige Information überhaupt zu einem menschlichen Lernenden zu transferieren, werden die Informationen über eine bekannte Syntax an den Lernenden mit Hilfe des Kommunikationsmediums übertragen. Das Wissen welches daraus vom Lernenden generiert wurde kann nicht direkt überprüft werden vom Lehrsystem, daher wird es mit Hilfe Übungen getestet. Es werden die Aktionen des Lernenden in Bezug auf das zu lehrende Wissen evaluiert um dann zu entscheiden, ob der Wissenstransfer zum Lernenden erfolgreich war.

Definition Pädagogikwissen

Die Eigenschaften, welche häufig als Grund für die Überlegenheit eines menschlichen Lehrers gegenüber einem Computersystem herangezogen wird, ist deren Adaptivität an den Lernenden und deren pädagogisches Wissen. Die professionelle Kompetenz umfasst bei Lehrern drei Wissensbereiche: Fachliches Wissen, fachdidaktisches Wissen und fächerübergreifendes pädagogisches Wissen [2009 KÖNIG and BLÖMEKE]. Fachliches Wissen entspricht dem oben definiertem Domänenwissen. Fachdidaktisches Wissen und pädagogisches Wissen beinhalten beide sowohl die Planung und Organisation, die Lehr- und Lernprozesse, als auch die Reflexion und Evaluation von Lehrinhalten. Der Unterschied liegt darin, ob dieses Wissen fächerunspezifisch angewendet werden kann oder nicht.

Die Unterteilung in pädagogisches Wissen und fachdidaktisches Wissen ist für die meisten Lehrprogramme unerheblich. Da es sich fast immer ausschließlich mit nur einem Fach beschäftigt, werden diese beiden in dieser Arbeit zusammengefasst zu Pädagogikwissen. Im Englischen wird fachdidaktisches Wissen als „pedagogical content knowledge“ und pädagogisches Wissen „educational enrichment rules“ bezeichnet. In Lehrprogrammen

wird auch der zusammenfassende Begriff „pedagogical knowledge“ genutzt und es wird auch kein hinreichender Grund gesehen diesen aufzuspalten.

Definition Softwarearchitektur

Es wurden viele Quellen analysiert um eine weit verbreitete und idealerweise allgemeingültige Definition einer Softwarearchitektur zu finden. Da es nicht „die eine“ Definition gibt, sondern Variationen über ein Thema, sind die wichtigsten Definitionen von Softwarearchitektur in der Tabelle R-II1a-5 aufgeführt. Diese und deren weitere Beschreibungen führen zu einer eigenen in dieser Arbeit verwendeten zusammenfassenden Definition und Beschreibung, die nachfolgend beschrieben ist.

Eine Softwarearchitektur ist ein abstraktes Modell des Aufbaus einer Software. Es werden die Entscheidungen für den Grundaufbau einer Software getroffen. Diese Entscheidungen sind ausschlaggebend für die Software und lassen sich nach finaler Entscheidung und Umsetzung als Software nur schwer ändern. Die Softwarearchitektur definiert die Komponenten und deren Schnittstellen. Das Verhalten der einzelnen Komponenten wird sowohl für jede einzelne Komponente selbst, als auch im Zusammenspiel spezifiziert. Die Komponenten werden in größere Subteile der Software zusammengefasst bis hin zu der Gesamtsoftware. Die Softwarearchitektur beschreibt einen Architekturstil als Leitlinie für die Organisation des Programms. Die Entscheidungen werden aufgrund mehrerer Faktoren getroffen, welche für das Softwareendprodukt relevant sind. Wie zum Beispiel die Benutzerfreundlichkeit, Performanz, Entwicklungszeit, Funktionsumfang, Robustheit, Systembeschränkungen, Entwicklungskosten und Wartbarkeit [2002 CLEMENTS].

Nicht jede Softwarearchitektur ist automatisch besser ist als eine ad hoc entwickelte Lösung. Allerdings entwickelt ein Softwarearchitekt an die gegebenen Umstände angepasst eine entsprechendSoftwarearchitektur, welche darauf ausgelegt ist Vorteile bei der Implementierung des Systems zu besitzen. Der Vorteil, Schlüsselentscheidungen zu identifizieren bevor die Implementierung beginnt, spiegeln sich dann unter anderem in den Entwicklungskosten und -zeit wider. Die zusätzlichen Kosten zur Entwicklung einer Softwarearchitektur fallen dabei nur einmal an. Denn eine Softwarearchitektur ist abstrakt genug um auf ein neues Projekt im gleichen Bereich übertragbar zu sein. Ein ähnliches neues Projekt, sei es auch nur teilweise, würde dann nicht von Grund auf neu entwickelt werden, sondern kann etwas aus der Softwarearchitektur von dem vorherigen Projekt verwerten. Ziel ist es, dass das Endprodukt so gebaut ist, dass die gewonnenen Erkenntnisse nicht versteckt im Programmcode sind sondern extrahiert wurden, dies verbessert auch die Langlebigkeit des ersten Projekts. Auch wenn nicht davon ausgegangen werden kann, dass die Erkenntnisse komplett losgelöst von einer groben Richtung der gewählten Implementierungsstrategie sind. So wird die Wahl des Programmierparadigmas Einfluss auf die Architektur haben.

Paper	Definition
[2011 BACHMANN, BASS and GARLAN]	„Each structure comprises software elements, relations among them, and properties of both elements and relations.“
[2010 MEDVIDOVIC and TAYLOR]	„The set of principal design decisions governing a system.“
[2003 BASS, CLEMENTS and KAZMAN]	„The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.“
[2000 HILLIARD]	„Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.“
[1999 BOOCH, RUMBAUGH and JACOBSON]	„An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architecture style that guides this organization—these elements and their interfaces, their collaborations, and their composition.“
[1995 SONI, NORD and HOFMEISTER]	„Software architectures describe how a system is decomposed into components, how these components are interconnected, and how they communicate and interact with each other.“
[1995 GACEK, ABD-ALLAH and BRADFORD]	„A collection of software and system components, connections, and constraints. A collection of system stakeholders' need statements. A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements.“
[1995 HAYES-ROTH, PFLEGER and LALANDA]	The architecture of a complex software system is its "style and method of design and construction".“
[1995 LAWSON, KIROVA and ROSSAK]	„[...] is typically defined in the context of the "requirements, design, implementation" sequence, referring the top level of the design stage, "...where the design issues involve overall association of systems capabilities with components." It also designates a higher level of abstraction, codification, and standardization, targeting the improvement of system design and making the complex system intellectually tractable.“
[1995 JACKSON and BOASSON]	„The definition of a set of generic component types together with: -a description of the properties of each type, -the rules governing the way each component type may interact with each other type -the style of interactions allowed between components, and -the rules which govern how a system (or subsystem) may be composed from instances of the generic components.“
[1995 ABOWD, ALLEN and GARLAN]	„Software architecture is an important level of description for software systems. At this level of abstraction key design issues include gross-level decompositional components, protocols of interaction between those components, global system properties (such as throughout and latency), and life-cycle issues (such as maintainability, extent of reuse, and platform independence).“
[1995 BOASSON]	„[...] "architecture" to mean a system structure that consists of active modules, a mechanism to allow interaction among these modules, and a set of rules that govern the interaction.“
[1994 CRISPEN and STUCKEY]	„An Architecture [...] consists of (a) a partitioning strategy and (b) a coordination strategy. The partitioning strategy leads to dividing the entire system in discrete, non-overlapping parts or components. The coordination strategy leads to explicitly defined interfaces between those parts.“
[1994 KRUCHTEN and THOMPSON]	„Software architecture deals with the design and implementation of the high-level structure of the software. It is the result of assembling a certain number of architectural elements in some well-chosen forms to satisfy the major functionality and performance requirements such as scalability and availability. Software architecture deals with abstraction, with decomposition and composition, with style and aesthetics.“
[1995 GARLAN and PERRY]	„A critical aspect of the design for any large software system is its gross structure that is, its high-level organization of computational elements and interactions between those elements. Broadly speaking, we refer to this as the software architectural level of design.“
[1994 KOGUT and CLEMENTS]	„Software architecture is loosely defined as the organizational structure of a software system including components, connections, constraints, and rationale.“
[1994 MORICONI and QIAN]	„A software architecture is represented using the following concepts: 1. Component: An object with independent existence, e.g., a module, process, procedure, or variable. 2. Interface: A typed object that is a logical point of interaction between a component and its environment. 3. Connector: A typed object relating interface points, components, or both. 4. Configuration: A collection of constraints that wire objects into a specific architecture. 5. Mapping: A relation between the vocabularies and the formulas of an abstract and a concrete architecture. The formula mapping is required because the two architectures can be written in different styles. 6. Architectural style: A style consists of a vocabulary of design elements, a set of well-formedness constraints that mush be satisfied by any architecture written in the style, and a semantic interpretation of the connectors.“
[1993 GARLAN and SHAW]	„[...] beyond the algorithms and data structures of the computation; designing and specifying the overall system structure emerges as a new kind of problem. Structural issues include gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements; physical distribution; composition of design elements; scaling and performance; and selection among design alternatives.“
[1992 PERRY and WOLF]	„We distinguish three different classes of architectural elements: processing elements; data elements; and connection elements. The processing elements are those components that supply the transformation on the data elements; the data elements are those that contain the information that is used and transformed; the connecting elements (which at times may be either processing or data elements, or both) are the glue that holds the different pieces of the architecture together.“
[1992 BHANSALI and NII]	„[...] topological organization of a set of parameterized modules, together with the inter-modular relationships. Designing a software system using a generic architecture consists of instantiating the parameters of each parameterized module by a concrete value while maintaining the inter-modular constraints.“
[1992 RECHTIN]	„The underlying structure of a system, such as a communication network, a neural network, a spacecraft, a computer, major software or an organization.“
[1990 LANE]	„Software architecture is the study of the large-scale structure and performance of software systems. [...] division of functions among system modules, the means of communication between modules, and the representation of shared information.“

Tabelle R-II1a-5: Definitionen von „Softwarearchitektur“ (Originalzitate)

In erster Linie sollte demnach eine Neuimplementierung eines konkreten Programms eine vorgegebene Struktur umsetzen, welche schon auf ihre Effizienz in verschiedenen Bereichen hin überprüft wurde. Dies ermöglicht es, schneller ein Design zu realisieren, da nicht erst mit Implementierungen ausprobiert wird ob es passt. Das Resultat kann Defizite in Bereichen aufweisen, an die während der Entwicklung nicht gedacht wurde. Diese Defizite stellen Risiken dar, ein Risiko welches mit geringerem Aufwand vor aufwändiger Implementierung vermieden werden kann. Eine Neuimplementierung dauert länger als ein abstraktes Modell. So ist es viel einfacher an einem Modell etwas zu ändern als in einer Implementierung. Dies ist umso aufwändiger, desto weiter die Programmierung vorangeschritten ist. Zudem entkoppelt eine Softwarearchitektur den Aufbau des Systems von der Implementierung. Damit wird auch die Kommunikation mit Personen, welche wenig oder kein Verständnis für die zugrundeliegende Software und Programmiersprache haben, erleichtert. Da die technischen Details unbehandelt bleiben, braucht es kein tiefgreifendes Verständnis, sondern es ist möglich, sich auf einer abstrakteren Ebene zu unterhalten ohne sich in Details zu verlieren. Durch die Erstellung eines grafischen Modells hat man auch eine visuelle, von der Informatikdomäne unabhängige Repräsentation der Spezifikation. Somit kann eine Kommunikation auch mit Experten von anderen Domänen verbessert werden. Wichtig ist dabei ein Wortschatz zu definieren, mit dem eine eindeutige Kommunikation der unterschiedlichen Sichten ermöglicht wird. Damit ist sichergestellt, dass kein Beteiligter vernachlässigt wird, weil das System mit einem Blick auf die verschiedenen Einflüsse entwickelt wird.

Zusammenfassend wird eine Softwarearchitektur hier folgend definiert:

- Entscheidungen wie etwas gebaut (aber nicht genau implementiert) wird
- Beschreibung der Elemente und deren Schnittstellen
- Verhalten der Zusammenarbeit der Elemente
- Zusammensetzung der Elemente zu Subsystemen und zu einem komplettem System
- Architekturstil um einen einheitlichen Stil und grobe Entscheidungen festzulegen

Zusammenfassend ist Softwarearchitektur für ein Lehrsystem aus folgenden Gründen sinnvoll:

- Identifiziert Schlüsselentscheidungen in der Entwicklung
- Ein Modell für die Risikoanalyse
- Abstrakte Modelle für die Visualisierung als ein Kommunikationsmittel mit Experten aus anderen Fachrichtungen (häufig in der ITS-Entwicklung, siehe Abschnitt „Beteiligte von ITS-Entwicklungen“ in dieser Arbeit)
- Ist zukunftssicherer und haltbarer als eine Implementierung

Definition Systemarchitektur

Der Begriff Softwarearchitektur sollte nicht mit einer Systemarchitektur verwechselt werden. Eine Softwarearchitektur konzentriert sich auf einen Themenbereich und die abstrakte Modellierung einer Struktur zum effektiven Aufbau einer Software zur Lösung eines größeren Problems. Es ist nicht die konkrete Implementierung der Lösung in einer Programmiersprache, sondern deren Funktionalität und Einteilung. Eine Systemarchitektur ist dagegen der strukturelle Aufbau der einzelnen verwendeten Software und Hardware. Es wird die Interaktion zwischen den Diensten über die Schnittstellen welche die Software und Hardware anbietet modelliert. Beide sind ein abstraktes Modell und beschreiben die Teile mit einem jeweiligen Aufgabenbereich und deren Beziehung zueinander. Bei größeren Projekten ist es meist ratsam, dass diese gemeinsam modelliert werden. Eine Softwarearchitektur beschreibt eine Software, während eine Systemarchitektur eine Sammlung von Software und Hardware beschreibt um größere Arbeitsabläufe (die nichts mit Programmieren zu tun haben müssen) zu bewältigen.

Definition Softwareframework

Der Begriff Softwareframework ist deutlich in verschiedenen Quellen als Quellcode definiert. Es ist entsprechend ein ausführbarer Code in einer Programmiersprache. Definitionen die diese Abgrenzung belegen sind in Tabelle R-II1a-6 aufgezählt. Abgekürzt wird das Softwareframework auch häufig nur „framework“ genannt. Die Definition von Softwareframework ist sehr stabil und kohärent.

Paper/Website	Definition
[2010 CURILEM, VIZCARRA and POO]	„A computer science framework can be summarized as a collection of something (e.g. classes, components, objects) which together define a pattern, template, or design of a system.„
[2009 MNKANDLA]	„A framework generally provides a skeletal abstraction of a solution to a number of problems that have some similarities.„
[2000 RIEHLE]	„The framework represents the cumulated experience of how the software architecture and its implementation for most applications in the domain should look like.„
[2018 SHARPENED PRODUCTIONS]	„A framework, or software framework, is a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform.„
[1997 SCHMIDT]	„A framework is a reusable, “semi-complete” application that can be specialized to produce custom applications„
[1994 GAMMA, JOHNSON and HELM]	„A framework is a set of cooperating classes that make up a reusable design for a specific class of software„
[2007 FERRARO]	„A framework is a real or conceptual structure intended to serve as guidance for the building of something that expands this structure into something useful.„
[2014 TRIPATHY and NAIK]	„A set of cooperating classes or frames that makes up a reusable design for a specific class of software.„

Tabelle R-II1a-6: Definitionen für „Softwareframework„ (Originalzitate)

Das Ziel eines Softwareframeworks ist die Wiederverwendbarkeit von Quellcode oder der Kompilierung und des implementierten Konzepts. Das Konzept definiert unter anderem ein oder mehrere Muster, Vorlagen oder Entscheidungen über die Implementierung von Softwareteilen. Es ist die Konkretisierung eines Konzepts wie einer Softwarearchitektur, bei dem bestimmte Teile absichtlich so gestaltet sind, dass sie leicht angepasst werden können. Es ist wie ein Skelett, das Gerüst einer resultierenden Software, von dem aus eine konkrete individuelle Version entwickelt werden soll.

Wichtig ist die Differenzierung von einem Softwareframework zu einer Softwarebibliothek (auch „software library“ oder nur Bibliothek genannt). Eine Softwarebibliothek ist nur eine Ansammlung von Funktionen. Die Kontrolle was und wann etwas aufgerufen wird bleibt bei dem Nutzer der Bibliothek. Ein Framework dreht dies um: die Kontrolle über Aufrufstrukturen und Prozessabläufe bleibt beim Framework. Diese Umkehrung wird „Inversion of control“ (IOC) genannt. Wenn die Kontrolle im Besitz des Frameworks ist, dann ist es sinnvoll und per Definition auch erforderlich, dass es ein Basisverhalten mit einer mindestens minimalen Funktionalität gibt. Die Bedienung eines Frameworks ist damit meist einfacher, da weniger Code ausreicht um komplexe, in dem Framework vordefinierte Abfolgen von Funktionen zu erzeugen. Damit es nicht bei diesem Basisverhalten eines Frameworks bleibt, muss es im Gegensatz zur Bibliothek auch erweiterbar sein. Damit hat man die Freiheit das Framework individuell zu konfigurieren und neues Verhalten zu integrieren, aber es ist schwieriger ein aus der Sicht des Frameworks falsches Verhalten zu Implementieren. Eine Bibliothek würde man eher für die Implementierung von einer Sammlung von Algorithmen und Helferfunktionen verwenden, welche unabhängig voneinander verwendet werden können.

II.1.b. Lerntheorien

Es gibt drei Kategorien von Lerntheorien, welche bis heute Bestand haben [2010 GLUCK, MERCADO and MYERS]. Historisch gesehen bauen sie aufeinander auf (siehe z.B. (Martens Diss)) und haben in der lern- und entwicklungspsychologischen Forschung nicht mehr alle unumstrittene Gültigkeit. Jedoch ist dies für die vorliegende Arbeit nicht von Belang, da bei der computerbasierten Lernsystementwicklung alle drei Kategorien nach wie vor (auch von Lernpsychologen) genutzt werden. Entsprechend unterscheiden sich die drei Kategorien stark voneinander um haben jeweils ihre Erklärung, wie Lernen funktioniert.

Behaviorismus

Der Behaviorismus ist das älteste der drei Modelle und geht aus der Perspektive der psychologischen Forschung noch auf eine Zeit zurück, in der wenig Wissen über das menschliche Gehirn existierte und keine Hirnstrommessungen (EEG) möglich waren. Es konnte daher aus Sicht der naturwissenschaftlich arbeitenden Psychologen nur mit dem gearbeitet werden, was sichtbar war: dies waren Reize und die Reaktion von Lebewesen auf Reize. Das Gehirn bleibt in diesem Geschehen eine Black Box.

Der Behaviorismus geht davon aus, dass auch der Mensch auf Reize reagiert. Eine Aktion ruft eine Reaktion hervor, siehe Abbildung R-II1b-1. Lernen ist dementsprechend nur ein automatisches Anpassen der Reaktion zu einem Reiz. Wenn man den Reizwiederholt wird, wird dadurch das starre Reaktionsverhalten abgespult und vernachlässigt stark die Bedürfnisse und Einflüsse eines Einzelnen. Unterschiedliche Personen müssten demnach mit genügend langem Training durch externe Rückmeldungen („Feedback“) bei gleichem Input, den gleichen Output generieren. Dabei reichen die natürlichen Reaktionen auf Reize nicht aus. Die natürliche Reize müssen überlagert werden können mit etwas, was beabsichtigt wird, wahrgenommen zu werden. Wenn die Reaktion auf diese gewollten wahrgenommenen Reize dann dauerhaft stimmt, dann wurde laut dieser Theorie erfolgreich gelernt. Es passt zu einem veralteten Modell des klassischem Unterrichts, indem ein positives oder negatives Feedback ausreicht um zu Lernen. Ein Lehrsystem wäre hier bloß ein Gehilfe für den einfachen Wissenstransfer, indem es Faktenwissen vermittelt durch Lehren in vorgefertigten didaktischen Abläufen wie Vorlesungen. Der Lernende wird dann durch Wiederholung und Richtig/Falsch-Bewertung trainiert sich dieses Wissen zu merken.

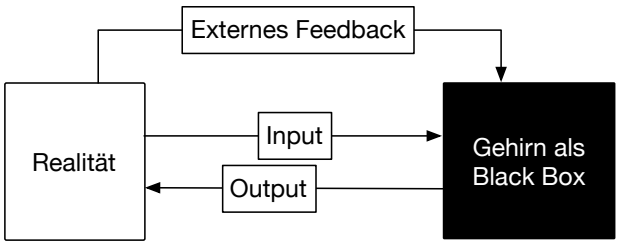


Abbildung R-II1b-1: Lernen aus Sicht des Behaviorismus

Konstruktivismus

Der Konstruktivismus, zeitlich dem Behaviorismus nachgeordnet, sieht das Gehirn nicht mehr als Black Box, sondern als selbstreferentielles System. Im Konstruktivismus wird davon ausgegangen, dass der Mensch ein individuell konturiertes Bild der Realität wahrnimmt und auch subjektiv im Gehirn verarbeitet. Die Reaktion des Menschen basiert damit auf seiner subjektiven Annahme über die Realität, welches durch subjektive Einstellungen verändert wurde. Wissen von anderen Menschen kann nicht einfach weitergegeben werden, sondern es muss vom Lernenden selbst konstruiert werden, siehe Abbildung R-II1b-2. Wenn das Gehirn ein relativ geschlossenes System ist, welches stark subjektiv beurteilt, dann muss auch das Lernen selbst explorativ gestaltet werden, um einen star-

ken Lerneffekt zu haben. Ein Lehrsystem wäre hier ein Betreuer, Trainer oder Coach der viele soziale Fertigkeiten besitzen muss um dem Lernenden zu helfen mit dem, im Ablauf wenig strukturiertem, Lehrmaterial zu interagieren. Das Lehrsystem als Betreuer, Trainer oder Coach würde dem Lernenden in komplexen Situationen beistehen, falls dieser nicht weiter weiß, ohne dabei die Antworten vorweg zu nehmen, sondern den Lernenden dazu bringen, die Lösung in seinem Gehirn zu konstruieren und nachträglich reflektierend auch in einem permanenten Abgleich zwischen Innenwelt und Aussenwelt zu handeln. Das Lehrsystem würde vielmehr kooperieren, anstatt das Lehrmaterial zu präsentieren. Er lässt den Lernenden dessen Vorstellungen ausprobieren und bewertet nachträglich.

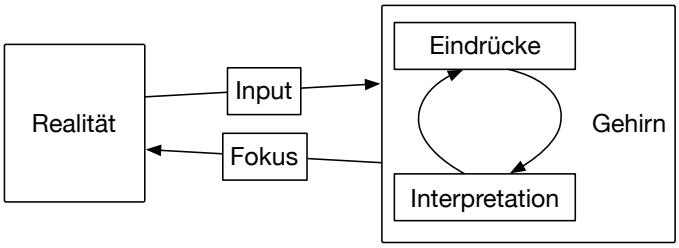


Abbildung R-II1b-2: Lernen aus Sicht des Konstruktivismus

Kognitivismus

Im Kognitivismus, der in der Gänze das behavioristische Paradigma ablöste und zu einem neuen psychologischen Paradigma wurde, wird das Gehirn als Sammlung kognitiver Prozesse mit erforscht. Innerpsychischen Prozesse werden analysiert. Der Kognitivismus ist die Lerntheorie der moderneren Psychologie. Das Gehirn kann die objektive Umwelt durch den Input, die einkommenden Daten, wahrnehmen und wie eine Maschine laufen dabei auf biologischer Basis Prozesse ab, siehe dazu Abbildung R-II1b-3. Diese haben verschiedenste Aufgaben, wie unter anderem die Sinneseindrücke zu verarbeiten, logische Entscheidungsfindung, kreative Ideen, Begriffsbildung, Muster wiedererkennen, Problemlösungsstrategien entwickeln oder das Mitteilen an andere Menschen. Das Verarbeiten von Input zu Wissen und umgekehrt von Wissen zu Daten passt zu den Wissenspyramide und dem Wissenstransfer (siehe Abschnitt „Grundlagendefinitionen“). Die Beurteilung von neuem Wissen und Entwicklung von neuen Problemlösungsstrategien mit der Generierung von abstrakterem Wissen entspricht dem Aufbau der SOLO Taxonomie (siehe Abschnitt „ Definition Solo-Taxonomie“ unter „Grundlagendefinitionen“). Das kognitive Lernen wird geprägt von dem Anbieten von nachvollziehbaren Lehrmaterial, mit dem Ziel beim Lernenden eine Einsicht zu erzeugen, warum das Domänenwissen genau so und nicht anders ist. Es soll ein Verständnis für das Domänenwissen entstehen. Das Lehrsystem ist in der Position des Tutors und im Dialog mit dem Lernenden, damit dieser die Verfahren, Algorithmen, Prozeduren und Wege lernen und verstehen kann, welche dazu da sind um eine Vielzahl an ähnlichen Problemen zu lösen (Transfer von Wissen). Dabei ist entscheidend dem Lernenden die Fähigkeit zu geben die richtige Auswahl an Methoden für eine konkrete Fragestellung zu wählen und seine Fertigkeit zu verbessern diese Methoden korrekt und zuverlässig anzuwenden.

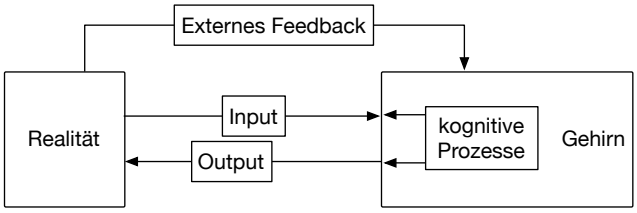


Abbildung R-II1b-3: Lernen aus Sicht des Kognitivismus

II.1.c. Lernmodelle

Zu den Lerntheorien gehören grundsätzlich auch die Lernmodelle, die für ein Lehrsystem sinnvoll sind, denn die Lernmodelle bauen auf den Lerntheorien auf. Eines oder mehrerer dieser Lernmodelle stellt die Grundlage für ein Lehrsystem dar.

Behaviorismus: Klassische Konditionierung nach Pawlow

Die klassische Konditionierung ist von dem Psychologen Pawlow (1849 bis 1936) entdeckt worden. Es zeigt, dass ein Reiz der bisher neutral von einem Menschen oder Tier wahrgenommen wurde nach einer Konditionierung eine bestimmte Reaktion auslösen kann. Die Konditionierung ist das Ausführen eines bedingten Reflexes zur gleichen Zeit, während mann einen unbedingten Reflex auslöst. Somit lernt der Körper diese beiden Reflexe zu assoziieren und reagiert auf den bisher neutralen bedingten Reflex so, als käme der unbedingte Reflex.

Behaviorismus: Operante Konditionierung nach Skinner

Entwickelt von Skinner (1904 bis 1990), welcher der Auffassung war, dass man Lernen nur auf objektiv beobachtbare Reize und deren Konsequenzen reduzieren kann. Er war eine wichtige Person für die Entwicklung eines linearen Lernprozess aus Lerneinheiten, dem programmierten Unterricht. Bei der operanten Konditionierung (auch instrumentelle Konditionierung genannt) erfolgt eine Verstärkung oder Abschwächung als Konsequenz ohne Verzug, nachdem ein Verhalten gezeigt wurde, um somit die Wiederholung des Verhaltens zu beeinflussen. Somit passieren Verhaltensweisen zukünftig häufiger, wenn diese eine positive Konsequenz für die Person haben.

Kognitivismus: Lernen am Modell nach Bandura

Entwickelt von Bandura (1925 bis heute). Er hatte die Theorie, dass Personen gedanklich Zukunftsszenarien entwickeln, um dann an der Realität zu überprüfen, ob das angenommene Modell stimmt. Damit lassen sich Verhaltensweisen, welche sehr komplex sind und nur durch Beobachtung bei anderen entstanden sind, erklären. Es werden resultierende Konsequenzen übernommen und damit erlernt ohne es selbst durchlebt zu haben.

Kognitivismus: Lernen durch Einsicht nach Köhler

Wie der Name schon sagt, braucht der Lernende in diesem Modell von Köhler eine Einsicht um plötzlich das Verständnis oder die Lösung zu einem Problem zu haben. Es ist ein Lernen, bei dem der Lernende ein Problem hat und dieses im Kopf neu strukturiert und so lange knobelt bis er eine Lösung gefunden hat. Beispielhaft ist das Lernen durch Problemlösung eine Art des Lernens durch Einsicht. Der Lernende kommt zu einer Einsicht indem er ein Problem löst. Dafür muss der aktuelle Zustand und das Ziel bekannt sein. Anstatt durch reines Ausprobieren versucht der Lernende das Problem durch Nachdenken zu lösen.

Konstruktivismus: Entdeckendes Lernen nach Ausubel

Entdeckendes (auch exploratives) Lernen [1974 AUSUBEL, NOVAK and HANESIAN] gibt dem Lernenden viel mehr Freiheiten und weniger Richtung vom Lehrer. Der Lernende selbst probiert Dinge aus und beobachtet. So versteht er durch von ihm bestimmte und durchgeführte Versuche Zusammenhänge und Regeln der Umgebung in der er sich bewegt und agiert.

Konstruktivismus: Wissensaufbau nach Piaget

Der Lernende hat bei Wissensaufbau die Aufgabe Wissen zu generieren. Besonders im Zusammenhang mit anderen Lernenden muss der Lernende sein Wissen zuordnen und organisieren, sodass es in ein (unter anderem von ihm selbst) entwickeltes Schema passt. Damit kann auch jeder Lernende das Wissen von anderen Lernenden aufgreifen und versuchen zu verstehen oder zu verbessern. Dazu wird von dem Lernenden ein hohes Maß an Selbstorganisation und Motivation erwartet.

II.2. Lehrsysteme und ITS

II.2.a. Digitalisierung der Lehrsysteme

Vorneweg sei erwähnt, dass wenn Lehrsysteme für einen Computer entwickelt werden, die Lehrmaterialien anstatt analog, auch alle digital vorhanden sind. Es reicht aber nicht ein analoges Vorgehen einfach auf einen Computer zu kopieren und dann von einer Digitalisierung zu reden. Es werden keine persönlichen Vorlesungen und Vorträge, Lernaktivitäten direkt mit anderen Personen oder Aufzeichnungen auf einer Kreidetafel, in Papierform oder Folien auf Overheadprojektoren verwendet. Es werden auch keine virtuellen Videokonferenzen verwendet auch wenn der Lehrer im gleichen Raum ist oder Videostreams von einem Kamerabild von Notizen den Lehrers. Die Ausgabe sind die Benutzerschnittstellen der Hardware, bei digitalen Systemen hauptsächlich der Bildschirm und die Lautsprecher als Ausgabe und die Tastatur und die Maus oder das Trackpad als Eingabe. Digitalisierung bedeutet mehr als nur den Inhalt auf einem Bildschirm anzuzeigen.

Digitalisierung bietet viele Vorteile. Zum Beispiel wurde ein signifikant höheren Lernzuwachs von Podcasts gegenüber Lehrbüchern festgestellt [2017 BACK, VON MALOTKY and SOSTMANN]. Der offensichtliche Vorteil der Digitalisierung von Lehrmaterial liegt in deren Organisation und Verwaltung. Der benötigte Platz und das Gewicht sind marginal klein. Die schnelle Suche, Kategorisierung und Annotierung mit Stichworten sind Funktionen, welche leicht zu implementieren sind und trotzdem einen großen Effekt bieten. Ein Lehrsystem hingegen mit analogen Lehrmaterial zu verbinden ist sehr aufwändig und bietet wenig Vorteile für den Lernenden. Von einer sehr simplen Digitalisierung, wie das bloße Einscannen der Lehrbücher unter Umständen noch ohne Texterkennung, ist abzuraten. Damit gehen die Vorteile von beiden Seiten verloren. Es fehlt sowohl die haptische Interaktion, die natürliche Beleuchtung, das Gefühl der Freiheit von elektronischen Geräten und Abhängigkeit von einer Batterie als auch existiert keinerlei Automatisierung und Massenspeicher eines digitalen Konsumgeräts.

Am Anfang der Geschichte der digitalen Lehrsysteme (ab circa 1950) wurde der Fehler gemacht, digitale Lehrsysteme genau so aufzubauen und zu verwenden wie es vom analogen Lehren bekannt war. Analoge Prozesse bloß auf einem Bildschirm zu übertragen, verschlechtert die Benutzerführung enorm (siehe „Scroller statt page turner“ [2004 MARTENS]). Solche - größtenteils älteren - Systeme lassen sich manchmal an den Skeumorphismen erkennen, zum Beispiel wenn der Hintergrund das Aussehen von gelben und leicht zerknitterten Papierseiten hat. Richtig digitalisierte Lehrmaterialen können zudem weitaus mehr Vorteile bieten als nur die Verwaltungsaufgaben (wie zum Beispiel Suchfunktion, kopieren/editieren/bereitstellen, unendliche Anmerkungen und Verlinkungen) zu vereinfachen, denn die digitalen Materialien sind leichter veränderbar und umwandelbar in andere Formate (zum Beispiel „Text2Speech“).

Digitale Lehrsysteme lassen sich grob durch die Einteilung von Briggs in eine von drei verschiedene Stufen einsortieren, siehe hierzu Abbildung R-II2a-1. Die erste Stufe geht davon aus, dass der Lehrer der Hüter des Wissens ist, der die Verantwortlichkeit erfüllt dies klar den Lernenden mitzuteilen. Der Wissenszuwachs eines Lernendes liegt allein an ihm. Es ist Schuld der Lernenden, wenn sie nicht effizient lernen. Wenn der Lernende etwas nicht versteht, liegt es daran, dass die Lernenden zu schlecht sind. Dies lässt sich auf digitale Lehrsysteme übertragen. Beispiel für die erste Stufe sind Systeme die die Last des Verstehens auf die Schultern des Lernenden legen, durch eigene Anmerkungen, Notizen und Übungen. Lernende sind Studenten oder Schüler, die das digitale Material entsprechend einfach zur Verfügung gestellt bekommen. Darunter fallen generell Materialsammlungen, Learning Management Systeme (LMS) oder virtuelle Universitäten.

Die zweite Stufe sieht den Lernerfolg im direkten Verhältnis zur Lehrfähigkeit des Lehrers. Die Kompetenzen des Lehrers stehen im Mittelpunkt. Lernende werden unterrichtet. Wie das geschieht entscheidet der Lehrer. Das Computerprogramm reflektiert eine einfache Form von Unterricht. Unterricht wird im Allgemeinen verbessert durch passende und motivierende Aufbereitung. Es gibt fast keine Unterstützung für die Lernenden. Dies wird in den digitalen Systemen durch computerbasierte Lehrsysteme widergespiegelt. Ein Beispiel für die zweite Stufe sind Game-Based-Learning-Spiele und Serious Games. Die dritte Stufe sind adaptive Systeme, welche sich daran anpassen was der Lernende macht und ihm die Möglichkeit geben aktiv zu beeinflussen wie gelehrt wird. Lernende werden vom Computer beobachtet und analysiert. Daran richtet sich die Form der Unterstützung die der Lernende benötigt. Beispiele sind intelligente Lehr-/Lernsysteme oder adaptive Hypermediasysteme.

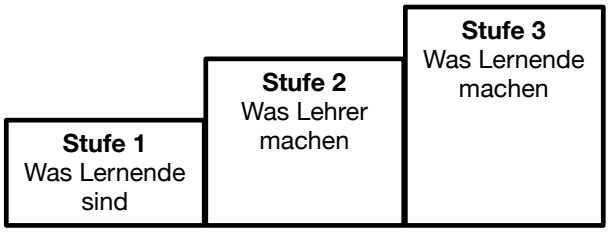


Abbildung R-II2a-1: Die drei Stufen von Briggs [2001 BIGGS]

Für die dritte Stufe gibt es natürlich auch Anforderungen. Den Lernenden muss das Ziel seiner Aufgabe im Detail deutlich sein. Die Anordnung der Lehrphasen mit den Übungsphasen müssen aufeinander aufbauen und sich auch abwechseln, damit die Lernenden das gelernte Wissen anwenden können und auch eine Rückmeldung über ihre Anwendung des Wissens bekommen. Die Reihenfolge welche empfehlenswert ist, besteht aus drei Dingen: Anzeigen des Lehrmaterials, Üben des Gelernten und dann eine Rückmeldung über die Qualität der Eingabe von dem Lernenden wieder an diesen geben. Zudem wird ein Art von Bewertungssystem empfohlen, welches es dem Lernenden erlaubt sich selbst einschätzen zu können.

Eine reine Notenverwaltung würde man heutzutage nicht mehr als ein Lehrsystem bezeichnen. In der klassischen Konditionierung ist eine Note genug um dem Lernenden ein positives oder negatives Feedback zu geben und damit als Lehrinstrument zu genügen. Ein Lehrsystem welches dann über eine reine Sammlung hinausgeht ist dann schon schnell an der unteren Grenze von Stufe zwei von den Briggs Stufen angelangt, allerdings ist der Schritt zur Stufe drei groß. Besonders die beiden höheren Stufen bauen aus Sicht eines Systems auf ein teilweises Verständnis des Inhalts von dem gespeichertem Lehrmaterial auf. Für die zweite Stufe ist es sehr vorteilhaft digitalisiertes Lehrmaterial zu haben, um nicht jegliche Zusatzinformation von Lehrmaterial digital extra zu erfassen. Für die dritte Stufe ist es praktisch notwendig Lehrmaterial in digitaler Form vorliegen zu haben, auch deshalb weil das Lehrmaterial sehr flexibel aus kleineren Komponenten besteht, welche passend zum Lernenden zu einer größeren Lehreinheit zusammengesetzt wird.

Lehrsysteme in dem Lerntheorien

Um zu entscheiden welche Lerntheorie für ein ITS am passensten ist, wird erst eine Einteilung von allen Lehrsystemen betrachtet. Abhängig von der Lerntheorie werden weitere Entscheidung für die Gestaltung der Konzepte für ITS gemacht. Hofmann teilt die Lehrsysteme in Kategorien abhängig von der Zugehörigkeit zu einer Lerntheorie ein, siehe dazu Abbildung R-II2a-2. Darauf folgend eine kurze Beschreibung von jeder Kategorie.

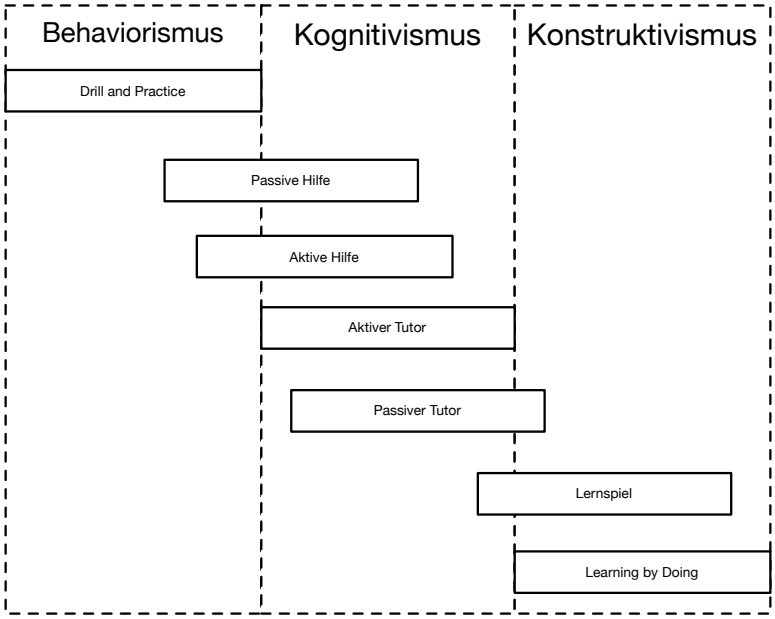


Abbildung R-II2a-2: Klassifizierung von Lehrsystemen [1997 Hofmann]

Die einsortierten Beispiele von Typen von Lehrsystemen für die verschiedenen Lernmodelle sind jeweils einer oder zwei Theorien zugeordnet. Hier aufgezeigt werden die folgenden.

- Drill and Practice: In solch einem System werden direkt die Übungen angezeigt und immer wiederholt bis alle perfekt beantwortet werden. Es ist eine Variante von den Lernmodellen der Konditionierung bei dem die exakt gleiche Frage irgendwann automatisch korrekt vom Lernenden beantwortet werden soll.
- Passive Hilfe: Hier ist eine Hilfefunktion für das Lösen und Lernen im Hintergrund vorhanden und wird nur wenn der Lernende aktiv danach verlangt angezeigt. Die Funktion zeigt dann für den aktuellen Lehrstoff relevante Zusatzinformationen. Diese sind nicht tiefer in die Materie, sondern entweder eine detaillierte Erklärung oder die dafür notwendigen Grundlagen verkürzt wiederholt oder zu diesen verlinkt.
- Aktive Hilfe: Hier ist eine Hilfefunktion wie bei der passiven Hilfe vorhanden, diese wird aber auch dann eingeblendet, wenn das Lehrsystem nach Beobachtung der Interaktion von dem Lernenden von einer Notwendigkeit der Anzeige der Hilfe ausgeht. Dabei wird für den aktuellen Lernkontext relevantes Lehrmaterial angezeigt. Das System kann dadurch mit Hilfe von bestimmten Parametern selbständig entscheiden wann der Nutzer eine Hilfe braucht.
- Aktiver Tutor: Genau wie bei einem passiven Tutor gibt es einen Prozess, welcher die Informationen anpasst, dieser kann aber mehr als Lehrmaterial anpassen, sondern zusätzlich werden auch aktiv Rückmeldungen zu den Interaktionen mit dem Lehrsystem gegeben. Anstatt nur passiv Wissen wie ein Monolog zu präsentieren, kann nun auch in Form von Übungen ein Dialog zwischen Lehrsystem und Lernenden stattfinden. Es ist damit auch möglich auf direkte Nachfragen des Lernenden einzugehen um dynamisch zu antworten.

- Passiver Tutor: Hier gibt es im Hintergrund einen Prozess welcher abhängig vom Verhalten des Lernenden die Präsentation und den Informationsgehalt der Lehrmaterials anpasst. Sodass dieser nur genau das angezeigt bekommt, was er für das Lernen benötigt. So werden zum Beispiel schon bekannte und sicheres Wissen weggelassen um genau diesen einen Lerner nicht zu langweilen.
- Lernspiel: Spielbasiertes Lernen bei dem der Lehrstoff nicht vordergründig ist sondern in einen anderen Ablauf integriert ist. Die Motivation des Lernenden weiter mit dem System zu interagieren, obwohl der Lehrstoff häufig trocken ist, ist hier ein ausschlaggebender Faktor. Somit wird die Zeit in der sich überhaupt mit dem Lehrstoff beschäftigt wird erhöht, indem es durch spielerische Weise vermittelt wird. Das eigentliche Ziel des Lernens mag dadurch in den Hintergrund treten.
- Learning by Doing: Hier ist der Lernende auf sich gestellt, denn die Lernumgebung gibt keinen Ablauf vor, sondern der Lernende kann durch eine gut gestaltete Umgebung ausprobieren und dadurch Erkenntnisse gewinnen. Das ist eine Form des entdeckenden Lernens, bei dem ein passiver Lehrinhalt durch die aktive Interaktion des Lernenden selbst herausgefunden werden muss.

Benutzerschnittstellen für die Interaktion zwischen dem Lernenden und dem System

Die Interaktionsmöglichkeiten sind bei einem System durch die Mensch-Maschine-Benutzerschnittstellen begrenzt und beschränken sich fast nur auf Tastatur und Maus/ Trackpad/Trackball als Eingabegerät und den Bildschirm und Lautsprechern als Ausgabegerät. Dennoch ist es ohne Probleme möglich auch andere Eingabegeräte, wie zum Beispiel aus der Spieleindustrie einzusetzen. Controller mit Beschleunigungssensoren, 3D-Kameras (Tiefenwahrnehmung) mit kombinierter Markierung der Nutzer könnten den Lehrsystemen durch weitere Eingabemethoden die Lerneffizienz erhöhen. In Zukunft werden wohl noch mehr Möglichkeiten zur Verfügung stehen, wie zum Beispiel Virtual Reality und Augmented Reality Brillen.

Kearsley [1987 KEARSLEY] unterscheidet fünf verschiedene Formen der Interaktion zwischen dem Lernendem und dem Lehrsystem für Softwaresysteme mit künstlicher Intelligenz betrachtet:

- Socratic dialogue: Dieser baut auf einen sichtbaren Dialog zwischen System und Lernenden auf. Sokrates fragt und der Lernende Antwortet. Wenn die Antwort falsch ist, stellt der Lehrende so lange angepasste Fragen, bis der Lerner selbst auf die Antwort gekommen ist [1993 KEARSLEY]. Fragen des Systems können den Lernenden herausfordern, motivieren mitzuarbeiten und helfen weiterzukommen, wenn sie auf das eingehen was kurz vorher der Inhalt des Lernenden war. Fragen des Lernenden sind schwer zu interpretieren, aber auch möglich.
- Coaching: Hier wird der Lernende unterstützt indem Vorschläge gemacht werden, was gelernt werden soll. Der Lernende besitzt weniger Entscheidungsgewalt, weil der Coach vorgibt und der Lernende nur zwischen wenigen Vorgaben wählen darf. Die Vorschläge vom System sollten dementsprechend gut sein um keinen Frust aufkommen zu lassen. Durch weniger Optionen im Lernpfad kann die Entwicklung auch wieder vereinfacht werden.
- Debugging: Es wird dabei auf die Fehler des Lernenden hingewiesen. Es soll verstanden werden warum der Fehler aufgetreten ist und wie man ihn vermeiden kann. Dabei gibt es beim Debugging zwei Varianten. Entweder die einfache Anzeige des Fehlers. Der Lerner muss dabei selbst lernen, wie man den Fehler dann beim nächsten Mal nicht macht. Oder Die Anzeige der Fehler und die Erläuterung, damit der Lerner versteht, was er falsch gemacht hat. Das wird durch ein Schritt-für-Schritt-Durchgehen der Auswertung erreicht.

- Microworlds: Es ist eine kleine Welt in der frei interagiert werden kann. Ein geschlossenes System, das durch eine Simulation (digitale Bedeutung) oder eine andere automatisierte Steuerung ablaufen kann und mit der man innerhalb der Systemgrenzen interagieren kann, welche der logischen Struktur eines kleinen Fachbereichs gerecht wird und diese widerspiegelt. Kleine Arenen in denen bestimmte Funktionen frei getestet werden können. Man lernt durch ausprobieren. Die Interaktion geht davon aus, dass der Lernende explorativ vorgeht, weil seine Neugierde groß genug ist, selbstständig neue Wissensbereiche in der Software zu entdecken.
- Explainable expert-systems: Diese Oberfläche ermöglicht eine Erklärung, warum die Logik des Lehrwissens in einem Fall genau so aufgebaut ist. Warum die Antwort des Lernenden zum Beispiel falsch ist, damit nachvollziehbar ist, warum die Bewertung so ausgefallen ist.

II.2.b. Geschichte der Lern-/Lehrsysteme hin zu ITS

Die geschichtliche Entwicklung von Lern-/Lehrsystemen ist relevant, da es in diesem Bereich sehr viele Begriffe gibt, welche unterschiedliche Definitionen haben. Über die Jahre hinweg gab es schon viele Versuche, den Computer für das Lernen und Lehren nutzbar zu machen, um eine Vision einer Zukunft mit automatisiertem Lernen zu erfüllen. Als Beispiel sei hier der computer-assisted „education room“ [1967 PHILCO-FORD CORPORATION] genannt. Zusammengefasst wurden die Computerartefakte für die Lehre unter verschiedenen Begriffen, am meisten werden allerdings die Retronyme „E-Learning“, bezeichnend für das elektronische Lernen, und „TEL“, als Akronym für „Technology Enhanced Learning“, verwendet. Folgend soll die Entwicklung der Lern-/Lehrsysteme von den Anfängen bis heute dargestellt werden. Um den Rahmen dieser Arbeit nicht zu sprengen, werden dabei nur Systeme und Konzepte betrachtet, welche relevant für die Entwicklung des „intelligenten Tutoring Systems“ sind. Auch wenn die Lehrsysteme immer komplexer werden, so bedingt dies nicht unbedingt eine kompliziertere Entwicklung eines einzelnen Systems, da die Werkzeuge und Konzepte auch ständig überarbeitet werden.

1930er

Die ersten Anfänge einen automatisierten Lehrer in der Forschung zu verwenden gab es in den 1920er Jahren. Echte Geräte in der Lehre einzubinden erst in den 1930er Jahren. 1930 kam eine Gerät auf den Markt mit dem Namen „The Automatic Teacher“ von S. Pressey [2004 PETRINA], siehe dazu auch Abbildung R-II2b-1. Es war eine Maschine, welche vorgefertigte Antwortbögen automatisch auswerten konnte, solange die Eingabe nur über diese Maschine erfolgte. Die Funktionsweise beruhte rein auf Mechanik. Der Ablauf war simpel und ähnelt im Grunde heutigen Single-Choice-Lernprogrammen, wenn auch auf analoger Basis. Es gibt eine festgelegte Anzahl an Fragen, welche genau fünf Antworten haben, davon ist genau eine korrekt. Die korrekten Antworten für die Fragen sind auf einer Papierrolle im inneren der Maschine durch Löcher markiert. Auf dem Gerät selbst gibt es einen Knopf pro Antwort. Es wird eine Frage als reinen Text in Papierform durch ein Fenster an dem Gerät angezeigt, die Antwort des Prüflings/Lernenden wird durch einen der Knöpfe abgegeben. Es gab mehrere Modi. Einmal den Prüfungsmodus, bei dem der Fragebogen die richtigen Antworten zusammenzählt, welche auf der Rückseite des Geräts sowohl angezeigt als auch gestempelt werden konnte. Dann gab es noch den Lernmodus, bei dem erst zur nächsten Frage gesprungen wurde, nachdem der Knopf für die richtige Antwort gedrückt wurde. Integriert war sogar ein Autorensystem, mit dem es möglich war die Lochkartenkodierung für bis zu 100 Fragen pro Rolle einzugeben.

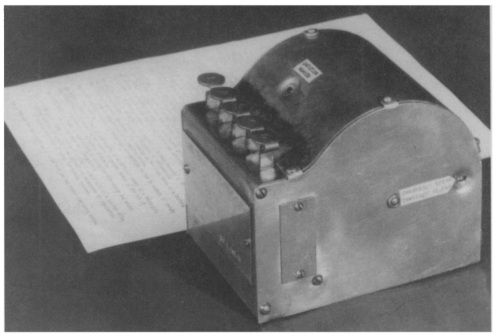


Abbildung R-II2b-1: Automatic Teacher [2004 PETRINA]

Das Gerät erschien mit einem Preis von damaligen $15, mit der Inflation sind dies $221 im Jahr 2018 [2018 IN2013DOLLARS]. Der Erfolg blieb allerdings aufgrund von mangelndem Interesse aus, auch aus Angst eine Maschine könnte Lehrer ersetzen. Effektives Lernen konnte damit nicht stattfinden, denn der Lernmodus war ohne Vorwissen nur ein raten.

1940er

Auch wenn E-Learning vorher skeptisch betrachtet wurde, war es aufgrund des Mangels an Lehrpersonal zwingend erforderlich die Lehre wenigstens teilweise zu automatisieren. Die USA hatte den Druck möglichst viele Soldaten während des zweiten Weltkriegs in kurzer Zeit auszubilden und realisierte es durch passive Lernvideos. Diese gehören zu den „Instructional Media“ [2001 REISER]. Instructional Media leisteten für die Weiterentwicklung der Lehrsysteme kaum einen Beitrag. Dennoch ist die Akzeptanz von Lehrmitteln mit technischer Bedienung und multimedialen Inhalten dadurch stark gestiegen. Zudem wurden kostspielige technische Geräte angeschafft. Insgesamt also eine Entwicklung der Einstellung der Lehrbeauftragten in Richtung einer guten Basis für den Einsatz von Lehrsystemen in der Zukunft. Dies unterstützte indirekt auch die Entwicklung von Lehrsystemen in der Forschung.

1950er

In den 50er Jahren wurden die ersten Computer in einer Größe entwickelt, welche kein eigenes Zimmer mehr benötigten, aber gleichzeitig eine Überprüfung von sehr einfachen Antworten eines Lernenden verarbeiten konnten. Es wäre also möglich gewesen, dass solche Computer in Lehreinrichtungen verwendet werden konnten. Der in Abbildung R-II2b-2 gezeigte LGP-30 kam im Jahr 1956 auf den Markt und erlaubte einfache Datenverarbeitung, welche für die automatisierte Lehre zu diesem Zeitpunkt ausreichend war. Mit seinem Preis von 50'000 US$ war es für die Lehre unbezahlbar. Dies entspricht einem Wert von $450'000 im Jahr 2018 [2018 IN2013DOLLARS]. Innovativ eingesetzte Lehrmittel zu dieser Zeit waren Kopfhörer und Rechenschieber. Dennoch zeigte es, dass Computer sich in eine Richtung entwickeln, dass in den nächsten Jahren ein bezahlbares koffergroßer Computer entwickelt werden würde, der das Lernen sowohl Zuhause, als auch in Lehreinrichtungen unterstützen würden.

Abbildung R-II2b-2: Erster Personal Computer [2018 COMPUTERHISTORY.ORG]

In den 1950ern war der Mangel an Lehrkräften in den USA besonders groß [2018 WEEBLY.COM], sie werden nicht umsonst „The Baby Boom“ genannt. Es wurde gerade in dieser Zeit von dem bekannten Behavioristen B. F. Skinner ein Lehrsystem ("teaching machine“) veröffentlicht. Von Skinner wurde ein Ansatz vertreten mit dem Namen operante Konditionierung ("operant conditioning“) und dieser basiert auf dem Behaviorismus. In diesem Ansatz folgt für jede Aktion-Reaktion-Kombination eine Konsequenz. Die theoretische Grundlage dahinter ist: Eine positive Konsequenz erhöht die Chance, dass der Proband das Verhalten wiederholt. Umgekehrt wird eine negative Konsequenz dazu führen, dass die Reaktion eher vermieden wird. Genau das bewirkt laut dem operanten Konditionierung das Lernen. Für eine Gruppe von Lernhardware die „teaching machine“ genannt wurde entwickelte er einen Prozess. Die simplere Struktur dieser „teaching machines“ führte mit seinem Vorschlag quasi dazu ihren gesamten Funktionsumfang festzulegen. Dem Lernenden wurden Aufgaben in Textform gestellt die er beantworten musste. Nur bei korrekter Antwort einer Frage war es ihm erlaubt zur nächsten Aufgabe voranzuschreiten. Es gab eine Anzeige ob die Antwort richtig oder falsch war. Genau diese Anzeige war die Konsequenz, welche für den Lernenden positiv oder negativ empfunden werden sollte. Die Antworten waren nicht vorgegeben, denn eines der Ziele von Skinner war es, dass die Lerner die Antworten nicht erkennen, sondern erinnern. Die Lerneinheiten sollten möglichst klein sein, damit die Konsequenz (zu belohnen oder zu bestrafen) der operanten Konditionierung auch häufig genug zum Einsatz kommt. Solche kleinen Lerneinheiten beschränkten sich auf kurze Absätze und Übungsaufgaben, welche auch „frames“ genannt wurden. Die operante Konditionierung war das Grundkonzept für eine Entwicklung die daraus entstand: Das für die Lehre von Skinner entwickelte Konzept „Programmiertes Lernen/Unterricht“ (im englischen „Programmed Instruction“ oder „Programmed Learning“ genannt, kurz PI) hatte drei essentielle Schritte [1991 FERNALD and JORDAN]. Der erste Schritt beschreibt den Aufbau des Lernstoffs. Er wird in kleine Lerneinheiten aufgeteilt, welche unabhängig voneinander in einer vorgegebenen Reihenfolge von dem Lernendem abgefragt werden (zum Beispiel eine Aussage oder eine Frage). Der zweite Schritt sagt aus, dass der Lernende diese beantworten muss, dies schließt mit ein, dass klar ist welche Antworten möglich sind. Als letzten Schritt kommt die Reaktion des Lehrsystems auf die Antwort des Lehrers ohne Zeitverzögerung. Dieses Konzept war sowohl für Maschinen als auch für spezielle vorbereitete Lehrbücher gedacht und hat die Vorteile, dass der Lernende die Lerngeschwindigkeit selbst bestimmen kann und zum Schluss der Lerneinheit eine Antwort vom System bekommt ob richtig geantwortet wurde, durch eine einfache richtig-falsch Auswertung.

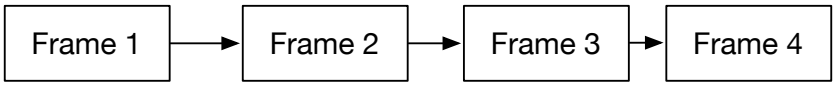


Abbildung R-II2b-3: Beispielanordnung der Lerneinheiten eines „Linear program“

Lineare Programme sind eine Konkretisierung des programmierten Lernens. Sie sind so aufgebaut, dass man sich von dem Start sequentiell über verschiedene Bereiche des Programmes durcharbeitet, siehe Abbildung R-II2b-3. Auf eine bestimmte Aufgabe folgte immer die gleiche nächste Aufgabe, bis zum Ende des Programms. Das forderte auch die Autoren (die Lehrer) der Systeme dazu auf ihren Lehrstoff sequentiell zu ordnen. Die Implementierungen des Konzepts in Computerprogrammen kamen zwar erst später, änderten aber nicht die Anforderungen. Es waren einfache Matheprogramme in denen man nur zur nächsten Aufgabe kam, nachdem die richtige Antwort als einfache Zahl eingegeben wurde. Eine häufige Implementation war auch ein Programm, welches aus Lückentexten, siehe hierzu R-II2b-4, bestand. Jede Lücke musste mit einem bestimmten Wort gefüllt werden musste.

An adjective can describe a quality of a noun as well as a pronoun. Therefore in these two sentences: "Sameer is regular in his work" and "He is funny too" the words regular and funny are _____ because they describe a quality of the noun Sameer and the pronoun He.

Tabelle R-II2b-4: Beispiellückentext für die „Programmed Instruction“ [2009 TIDDLYSPOT.COM]

Auch wenn sich Psychologen von der Theorie von Skinner als Lernmethode im Allgemeinen abgewendet haben, so hat Skinner einen wesentlichen Einfluss in der Entwicklung von Lehrsystemen gehabt, indem er die Verwendung von Computern als Lernmittel stark verbreitet hat. Und auch heute wird programmiertes Lernen angewendet. Zum Beispiel funktionieren Sprachlabore noch immer auf dem gleichen Prinzip, welches Skinner etabliert hat.

Es gab auch noch andere Varianten der linearen Programme. Sydney L. Pressey benutzte nur Single-Choice-Fragen bei jeder Lerneinheit für seine „teaching machines“ als Weiterentwicklung des Automatic Teachers. John Barlow entwickelte eine Art des Linearen Programms mit dem Namen „conventional chaining“ bei dem jede Lerneinheit mit der zweiten verknüpft sind für eine Rückkopplung. Weitere waren die RULEG und EGRULEG Konzepte [1976 VAN HOUT and METTES] mit jeweils kleinen Änderungen an den Verknüpfungen zwischen den Lerneinheiten, welche schon als einen Wunsch angesehen werden konnten, die Linearen Programme zu verbessern. Es benötigte schon eine tiefgreifende Änderung an dem Konzept um es wirklich zu verbessern.

1960er

„Branching programs“ (Verzweigte Programme) sind eine weitere Form des programmierten Lernens, welche mehr Möglichkeiten bietet und adaptiver sind als die linearen Programme. Crowder [1959 CROWDER and GALANTER] und auch Pressey [1963 PRESSEY] fanden, dass die linearen Programme einen konzeptionellen Fehler besaßen. Sie waren zu starr. Crowder wollte seinen Prozess als Ausbilder bei der U.S. Air Force abstrahieren, während Pressey nicht wollte, dass ein Lernender mit falschen Annahmen zu weiterführenden Lehrmaterial gelangt und damit die Fehlerrate unnötig negativ beeinflusst. Die linearen Programme aber behandelten gewollt jeden Lerner stetig gleich. Die Grundlage war schließlich der Behaviorismus, welcher Lernen als eine vom Lernenden unabhängige und sequentielle Aktivität ansah. Mit linearen Programmen zu lernen hat eine abnehmende Lerneffizienz, weil es zu Wiederholungen von bereits gefestigtem Wissen kommt. Laut Skinner sei es die Schuld des Lehrstoffs des linearen Programms, falls der Lernende versagt, nicht die des Lernenden oder des Konzeptes. Um ein Programm an die Leistungen der Zielgruppe anzupassen, konnten aufgrund ihrer sequentiellen Natur keine linearen Programme verwendet werden. Dabei war es egal ob der Lernende eine gleichbleibend hohe Konzentration während der Benutzung hatte, oder dass der Einzelne unterschiedliche Interessen, Vorwissen und Schwächen besitzt. Carbonell sah sogar kaum einen Mehrwert in einem linearen Programm gegenüber einem Textbuch [1970 CARBONELL]. Es war für ihn eher maschinenähnliches Lernen. Diese Nachteile sollten durch ein „Branching Program“ verhindert werden, denn die Antwort des Lernenden wurden berücksichtigt und ein Lerner konnte nicht nur durch korrekte, sondern auch durch falsche Antworten voran kommen. Der Aufwand für das Erstellen eines „Branching Program“ war durch seine höhere Komplexität durch mehr Inhalt für Lernende die falsch geantwortet haben auch größer. Es wurden speziell für die Autoren „Domain Specific Languages“ (DSL) geschaffen, welche ihnen die Strukturierung der einzelnen Lerneinheiten ermöglicht. Im Vergleich zum „Linear Program“ bestimmten die Antworten des Lernenden einen von mehreren möglichen Wegen durch das „Branching Programm“ zu gehen und damit hatte es einen erheblichen Vorteil gegenüber Lehrbüchern oder Lehrvideos: Es besaß keinen starren

Ablauf. Antworten konnten zudem durch „Pattern Matching“ mehr als nur richtig oder falsch sein. Anstatt das eine Antwort in die Kategorien richtig und falsch eingeteilt werden, wurde ihre Passgenauigkeit zu den definierten Mustern bestimmt, bei dem jedes Muster zu einer Aktion führen kann. Es wurden einfache Regeln für Programme für die Adaption des Schwierigkeitsgrads an den Lernenden veröffentlicht [1969 Uhr].

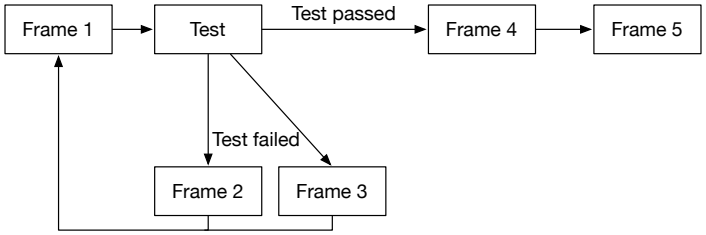


Abbildung R-II2b-5: Beispielaufbau eines „Backward Branching Program“

Ein „Branching Program“ diagnostiziert anhand von Single-Choice-Fragen , ob der Lerner die vorherige Lerneinheit ausreichend verstanden hat. Der Ablauf besteht aus sich vier in dieser Reihenfolge wiederholenden Teilschritten:

- Lernen: Das Domänenwissen wird didaktisch ansprechend präsentiert und es wird von dem Lernenden erwartet, dass dieser versucht es zu verstehen.
- Antworten: Der Lernende wird dann einem Test (eine Single-Choice-Frage) unterzogen, um festzustellen, ob er das zu lernende Fachwissen erlernt hat.
- Diagnose: Hier ist die Verzweigung und der wesentliche Unterschied zu einem linearen Programm. Der Lernende kann anhand der Antwort zu unterschiedlichen Lerneinheiten weitergeleitet werden.
- Ermutigung: Am Anfang der nächsten Lerneinheit wird die korrekte Antwort bestätigt und der Lerner gelobt.

Ein „Branching Program“ hat damit über das gesamte Programm eine mögliche Fehlerdatenbank integriert. Es gibt zwei Methoden der Verzweigung. Es war Crowder wichtig, dass ein „Branching Program“ flexibel genug ist, um dem Lernenden bei einer falschen Antwort an einen Frame weiterzuleiten, welcher die Erklärung liefert, warum die Antwort falsch war. Deshalb werden beide Methoden genau dies tun. Das „Backward Branching“, wie in Abbildung R-II2b-5 zu sehen, stellt die einfachere Variante dar. Hier wird der Lerner nach der Erklärung des Fehlers wieder zu der Lerneinheit weiterleiten, welche zu dem durchgefallenen Test passt. Also eine Wiederholung der letzten Lerneinheit solange bis der Lerner die richtige Antwort gibt. Richtige Antworten führen sofort zur nächst komplexeren Lerneinheit. Die bessere und aufwändigere Variante ist das „Forward Branching“, welches verhindern will, dass ein Lerner immer wieder die gleiche Frage gestellt bekommt und deshalb entweder direkt nach der Erklärung des Fehlers zur nächsten Lerneinheit weiterleitet oder eine oder mehrere weitere ähnliche Fragen zu der Lerneinheit stellt, bis eine richtig gemacht wurde.

Beispiele des „Branching Program“ sind Lernprogramme für Single-Choice Fragen. Besonders Single-Choice-Fragebögen (wie sie in Strukturen von Staatsexamina vorkommen) sind dazu ideal, weil sie aufgrund der hohen Anzahl an Lernern wirtschaftliches Potential haben und deren Struktur einfach und einheitlich ist, zum Beispiel dass die Anzahl der Antworten immer fünf ist. Es gibt zum Beispiel viele, auch auf mobilen Endgeräten, Programme für das üben von den unterschiedlichen Führerscheinen oder der juristischen und medizinischen Examina. Erst wird der Inhalt gezeigt, dann eine Frage gestellt und die Auswahl einer Antwort wird abhängig von dieser kommentiert.

1970er

1972 wurde die Programmiersprache „C“ veröffentlicht, welche damals wie heute eine der drei meisten verbreiteten Programmiersprachen ist. Schon 1986 war die Programmiersprache „C“ laut TIOBE-Index [2016 TIOBE.COM] die beliebteste Programmiersprache und bleibt bisher in den Top zwei. Damit zeigt sich das Grundbausteine von Software, die zu dieser Zeit entwickelt wurden, bis heute gültig sind. So auch bei den Lehrsystemen. Ende der 60er Jahre aber hauptsächlich in den 70er Jahren gab es einen großen Umschwung in der Entwicklung von einfachen Lehrsystemen hin zu „Computer Assisted Instruction"/„Computer Aided Instruction“ (CAI) Systemen. In Europa wurde der Begriff „Computer Assisted Learning“ (CAL) verwendet, als auch seltener die Begriffe „Computer Based Instruction“ (CBI), „Computer Based Education“ (CBE), „Computer Managed Instruction“ (CMI) oder „Computer Enriched Instruction“ (CEI). Durch die damalige weitere starke Verbreitung von Computern wurden diese auch im Lehrbereich zunehmend eingesetzt. Lehrsysteme wurden offener und vielseitiger durch die Entwicklung des Heimcomputers (im englischen „Personal Computer“ abgekürzt PC). Es gab mehr Forschung in die unterschiedlichsten Richtungen wie ein Lehrsystem aufgebaut werden soll. Es entstanden etliche Parallelentwicklungen - oft die gleiche Technik mit unterschiedlichem Namen. Dies merkt man auch anhand der vielen unterschiedlichen Namen für ein CAI-System. Der Begriff des CAI-System wurde allerdings für viele verschiedene Systeme verwendet, welche sich im Konzept stark unterschieden. Damit ist ein CAI-System im Konzept diffuser als „Linear Programs“ oder „Branching Programs“. Anfänglich waren die CAI-Systeme Kabinen, jeweils für einen Lerner mit einer Tastatur, einem Fernseher und dem Computer. Wie schon in den Abkürzungen zu sehen, ist der Computer ein zentraler Aspekt eines CAI-Systems. CAI stellt damit als erstes ein interaktives Lehrmaterial dar, welches durch einen Computer lehrt. Es zeigt das Lehrmaterial, beobachtet das Verhalten des Lernenden und vergleicht es mit einer Referenzlösung um dann eine Rückmeldung zu geben, siehe dazu Abbildung R-II2b-6 in denen Rechtecke Programmaktionen sind („program actions“) und die Ovale vorher definiertes Programmwissen („canned program knowledge“) sind.

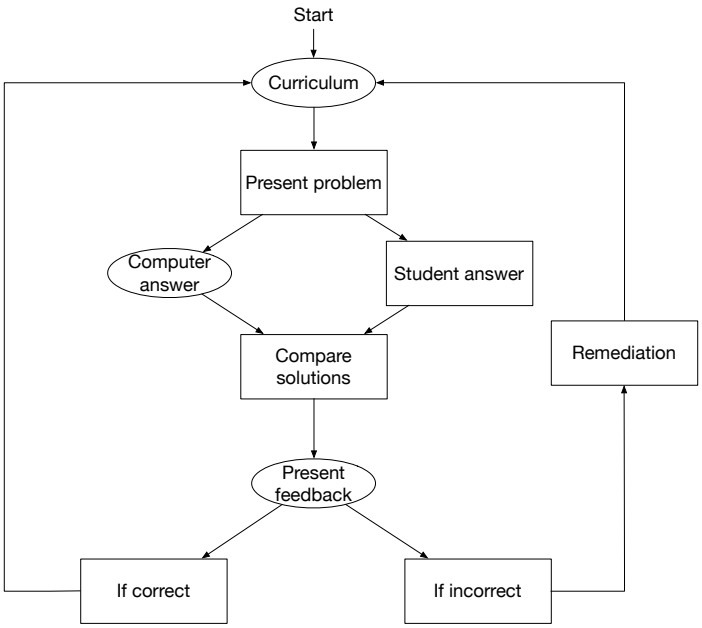


Abbildung R-II2b-6: Aktivitätsdiagramm eines CAI-Systems [1994 SHUTE and PSOTKA]

Es gibt verschiedene Methoden mit einem CAI-System zu lehren die über die Jahre entstanden sind. Durch die vielen interaktiven und medialen Möglichkeiten Lehrstoff an die Lerner zu vermitteln ist die Frage nach welchen Maßstäben man eine Vorgehensweise wählen sollte. Wenn man die finanziellen und personellen Aspekte außen vor lässt, ist eine

erhöhte Lerngeschwindigkeit das Ziel. Wie schon von anderen wissenschaftlichen Arbeiten bestätigt, hat mediales Lehrmaterial durchaus die Möglichkeit mehr in kürzerer Zeit zu lernen [2017 BACK, VON MALOTKY and SOSTMANN]. Ein CAI-System muss nicht lokal auf dem Computer laufen, es kann auch der Webserver sein, der die Funktionalität bereit stellt.

Beispiele heutiger CAI-Systeme sind „Coolmathkids“ [2018 COOLMATH4KIDS], „AplusMath „[2018 VARSITY TUTORS] und Inspiration [2018 INSPIRATION SOFTWARE INC]. Einfache CAI-Systeme sind hingegen keine große Weiterentwicklung zu einem „Branching Program“. Schon aus Kostengründen wurden Computer für Aufgaben eingesetzt die vorher analoge Maschinen gemacht haben. Deshalb gab es einen speziellen Begriff für eine bestimmte Art von CAI-Systemen welche eine wirkliche Neuerung zu einem „Branching Program“ darstellt, die „Generative CAI“. Diese waren die ersten adaptiven Lehrsysteme. Mit ihnen war es möglich Lerninhalte zu erzeugen, anstatt nur aus schon vorhandenen Lernmaterialien passende auszuwählen. Daher auch der Begriff „Generative“ im Namen, von dem Wort „generieren“ aus dem Englischen. Problemstellungen konnten angepasster sein ohne eine riesige Datenbank an vorgefertigten Fragen zu haben. Anstatt die Lehrmaterialen und Tests alle von einem Menschen vorher zu erstellen, war es in einigen Domänen mit klaren Definitionen und Regeln möglich, all diese von dem System generieren und überprüfen zu lassen. Mathematik und Physik sind hier besonders passend. Adaptiv war das System in der Hinsicht, dass die generierten Lehrmaterialen sich in dem Schwierigkeitsgrad an die Fähigkeiten des Lernenden anpassen konnten.

1980er

Das zentrale Problem bei den Lehrsystemen bis zu diesem Zeitpunkt ist die geringe Adaption des Systems an den Lerner, Fragen und Antworten wurden nicht spezifisch an die Fähigkeiten des Lernenden angepasst. Sein Wissensstand wurde nur sehr simpel modelliert. Meistens wurden nur wenige oder gar nur ein Zahlenwert verwendet um das Wissen des Lernenden zu modellieren.

Es gab in der Forschung eine Integration der Fortschritte der Künstlichen Intelligenz in die psychologische Lehrforschung. Vor allem hatten Psychologen dabei die Idee, ein beobachterfreies und damit beobachtbares Lehr-/Lernumfeld zu schaffen, um den Lernenden beim Wissenserwerb optimal beobachten zu können (Quelle Anderson). Daraus entstand eine neue Form von Lehr- und Lernprogrammen, welche adaptiv zum Verhalten des Nutzers mehr als nur das Lehrmaterial anpasste, indem sie ein Lernermodell zur Laufzeit erschufen und präzisierten. Diese Systeme wurden Intelligente Tutoring Systeme (ITS) genannt. Anfangs hatten sie noch den Namen „Intelligent Computer Assisted Instruction“ (ICAI) in Erweiterung der oben genannten CAI. Der Begriff ITS hat allerdings den des ICAI abgelöst. Intelligent kann hier irreführend sein. Gemeint ist hier keine menschliche Intelligenz (im Sinne des psychologischen Intelligenzbegriffes). ITS weiß welche Daten es lehren soll, und auch wie und wem. Es wird didaktisches Wissen über das Lehrmaterial als auch Wissen über den aktuellen Lerner verwendet. Der Wandel von einem ICAI-System der ersten Generationen mit eher elementaren Funktionen eines „intelligenten“ Systems zu einem autonomen ICAI welches mit einem ITS gleichzusetzen ist, ging schleichend voran und so ist auch keine klare Begriffliche Trennung vorhanden. Die Freiheiten der Forscher in dieser Bereich tragen auch dazu bei, dass viele Begrifflichkeiten entstehen, welche sich stark ähneln. Es ist schwer zwischen den vielen unterschiedlichen Begriffen zu unterscheiden, wenn diese teilweise nur marginal verschieden sind. Darunter leidet auch die klare Definition der ITSs. Dies ist keineswegs außergewöhnlich, denn ähnlich erging es auch einigen anderen Konzepten in der Entwicklung von Programmen. Als Beispiel sei hier das extrem erfolgreiche Model-View-Controller-Pattern genannt, welches bis heute stark unterschiedliche Definitionen hat, das MVC-Pattern von Apple heißt bei

Microsoft Model-View-Presenter und das MVC-Pattern ist für Microsoft etwas anderes. Das sehr erfolgreiche Pattern gibt es für Webapps auch unter dem Namen ELM.

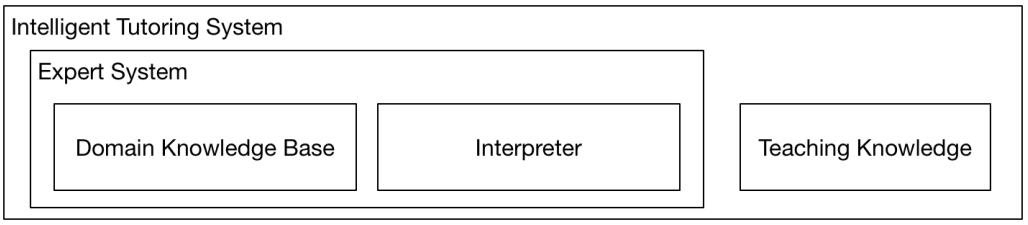


Abbildung R-II2b-7: Erste Unterteilung in Komponenten [1984 CLANCEY]

Das Konzept des ITS wurde zunächst abstrakt und einfach definiert, — vor allem, weil die Software noch nicht so komplex war. Das erste ITS-Konzept ist von William J. Clancey [1984 CLANCEY] und in Abbildung R-II2b-7 zu sehen. Es ist die erste ITS-Softwarearchitektur in der Forschung. Er beschreibt darin die Komponenten welche notwendig sind um ein ITS zu entwickeln. Die Beschreibung des ITS durch die Komponenten wurde bis heute als Basis beibehalten und weiterentwickelt.

Die rasante Einbindung von Künstlicher Intelligenz in Lehrsystemen kam durch das Forschungsinteresse von Psychologen. Der Durchbruch des PCs in den Massenmarkt führte zwar zu Computers in privaten Haushalten, doch waren die zu der Zeit aktuellen ITS zu anspruchsvoll in ihren Systemanforderungen und in den Entwicklungskosten. In diesem Jahrzehnt hat der Personal Computer die Haushalte in den USA erreicht. Mit dem überaus erfolgreichen „IBM PC“ (1981) und dem „Apple Macintosh“ (1984) kamen für Einzelhaushalte bezahlbare Computer mit einer für einfache Lehraufgaben guten Rechenleistung und Größe auf den Markt, welche auch eine gute Benutzerfreundlichkeit aufwiesen.

Die großen Unterschiede von ITSs zu den CAI-Systemen werden folgend aufgezählt:

- Die gute Kommunikation von ITSs. ITSs ist es möglich sich in ihrem, wenn auch manchmal kleinem, abgesteckten Domäne Fachwissen und Probleme dem Lernenden darzustellen. Auch wenn die ITSs dieser Zeit sprachlos waren [1984 CLANCEY] (Zitat: „At a certain level, MYCIN is aphasic - able to perform, but unable to talk about what it knows“) wurden mit der Zeit die aussagekräftigen Rückmeldungen des ITS immer mehr eine Eigenschaft von ITSs
- Das ITS hält ein Modell des Wissens des Lernenden, das Lernermodell. Dieses Modell beinhaltet getätigte Aktionen von ihm, welche wiederum genutzt werden kann um das Verhalten des ITS zu beeinflussen
- Der Lerner kann mit dem ITS in einem komplexeren Rahmen interagieren. Anstatt nur Lektionen auszuwählen, kann das ITS (vorgefertigte) Aktionen bewerten und antworten
- Es gibt beim ITS keine kleinteilige vorgefertigte Sequenzen an Lehrmaterial. Zwar kann Lehrmaterial aufeinander aufbauen, allerdings ist die Reihenfolge jeder Aktion nicht festgelegt

1990er

Gegen Ende der 90er begannen einige Universitäten damit Learning Management Systeme (LMS) zu verwenden. Dies war ein großer Umbruch in der Lehre, weil die Lehrpersonen begannen mehr und mehr digitale Lehrmittel zu verwenden. Ein LMS beinhaltet neben reinem Lehrmaterial auch die Funktionen für das Verwalten. Zudem kann man auch weitere organisatorische Aufgaben an ein LMS abgeben, wie zum Beispiel die Notenverwaltung, Verwaltung von Übungszetteln und deren Abgabe und Benotung, personalisierte Ansichten oder Kommunikation mit den Lernenden. Historisch gesehen war der Begriff LMS nur der Teil eines „integrated learning systems“ (ILS) namens PLATO K-12 [2005 BAUSERMAN, CASSADY and SMITH], welcher die Managementfunktionen abdeckte.

Zwar war dies keine Innovation in der automatisierten Lehre, dennoch hatte es Vorteile für diese, denn es stellte einen sehr großen Schritt für die Akzeptanz von Technologie in der Lehre dar. Dies bedeutet auch einen sehr großen Anstieg an digitalem Lehrmaterial. Beides ist positiv für die Einbindung von ITSs in der Lehrorganisation von Schulen, Hochschulen und Fachschulen. Der Unterschied von LMS zu „Course Management System“ (CMS), nicht zu verwechseln mit einem „Content Management System“, welches in der Webentwicklung als zentrale Oberfläche häufig verwendet wird. Ein CMS ist hauptsächlich ein Organisationswerkzeug um Lehrmaterialien bereit zu stellen, Lernende Kursen zuzuordnen, Leistungen zu erfassen, eine Kommunikation zwischen Lehrer und Lernender bereit zu stellen (um zum Beispiel organisatorische Informationen dem Lernenden mitzuteilen) oder auch eine Kommunikation zwischen den Lernenden in Form eines Forums oder auch nur um digitale Abgaben von Übungsaufgaben dem Lernenden zukommen zu lassen. Ein LMS hingegen verlagert mehr als nur die Organisation eines Kurses in die digitale Welt und das Internet. Es hat Funktionalitäten die es dem Lernenden erlauben durch Interaktion zu Lernen, anstatt sich ein digitales Buch anzuschauen.

In den 90er Jahren hatte das Internet und seine wichtigste Anwendung, das „World Wide Web“, seinen Durchbruch [2010 BRAUN]. Durch die Verlagerung des Lehrmaterials auf das Internet, gab es viele neue — sogenannte webbasierte — Softwaresysteme für die Lehre [1998a BRUSILOVSKY]. Es entstanden Neubezeichnungen von CAI-Systemen, indem „Computer“ durch „Web“ ersetzt wurde. Es entstand „Web Based Training“ (WBT), „Web Based Instruction“ (WBI), „Web Based Education“ (WBE) und „Web Based Learning“ (WBL). Das Konzept blieb gleich, mit einem Unterschied: WBT ermöglichtem es dem Lernenden zu lernen, ohne Lehrmaterialen auf seinem Computer zu haben. Auch der Begriff des E-Learning ist erst im Jahr 1999 entstanden und fasst alle technologischen Lehrsysteme unter diesem Begriff zusammen. Eines der Vorteile des Internets war eine zentrale Verwaltung der Lehrmittel und Anwendungen, wodurch diese leicht aktualisiert werden konnten. WBT beruhten meist auf dem Konzept der Programmed Instruction. Auch wenn ein Hype um E-Learning entstand, welcher vorteilhaft für die Reputation der ITSs war, so wurden adaptive Systeme nicht breitflächig eingesetzt.

Andere Bereiche der Informatik wurden weiterentwickelt, welche auch einen Einfluss auf die Entwicklung von ITSs hatten, darunter vor allem das Softwareengineering. Die Design Patterns als Konzepte zur Lösung von wiederkehrenden Problemen in der Programmierung wurden durch das sehr erfolgreiche Buch „Design Patterns: Elements of Reusable Object-Oriented Software“ [1994 GAMMA, JOHNSON and HELM] populär. Eine neuentwickelte Softwarearchitektur für die Entwicklung eines ITS wurde in solch einer Zeit besonders gern gesehen. Das ITS-Konzept entwickelte sich zu einer groben Softwarearchitektur und integrierte das „student knowledge module“, eine neue Komponente welche das bisher erworbene Wissen des Studenten darstellt, siehe hierzu Abbildung R-II2b-8. Einfach implementiert ist es als Overlay-Modell nur eine Submenge von dem zu unterrichtendem Fachwissen. Die Softwarearchitektur mit kleinen Abweichungen hat bis heute Bestand als der Grundaufbau eines ITS. Es wurde ein Wechsel von CAI zu ITS erwartet und damit kamen auch Bedenken für den höheren Entwicklungsaufwand und wie man ihn mit Autorenwerkzeugen verringert könnte [1996 MURRAY].

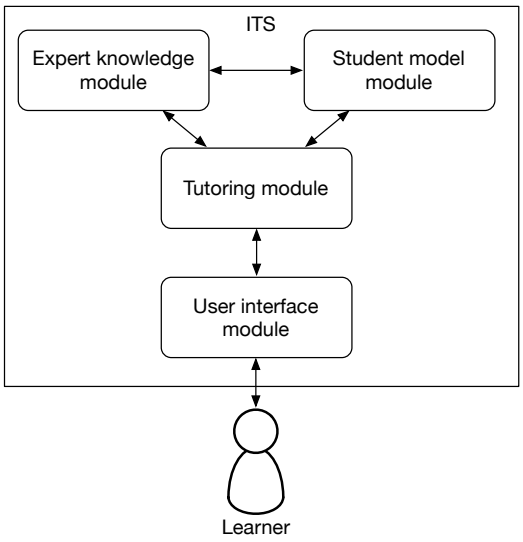


Abbildung R-II2b-8: Allgemeiner Aufbau eines ITS [1990 NWANA]

Es wurde erkannt, dass die Lehrinformation mehr enthalten muss als es bisher der Fall war, das Wissen um die Didaktik [1996 STANKOV], welche die didaktischen Lehrinformationen in einer eigenen Komponente verwaltet, siehe Abbildung R-II2b-9. Dieses Didaktikwissen sollte nicht als ein Teil des Domänenwissens gespeichert werden. Eine Trennung zwischen Domänenwissen und pädagogischem Wissen gibt der Software zwei klar getrennte Bereiche aus denen eine Lehreinheit zusammengesetzt werden kann. Es ermöglicht dem Lehrsystem auch, pädagogische Schlussfolgerungen zu ziehen, welche unabhängig von Lehreinheiten sind.

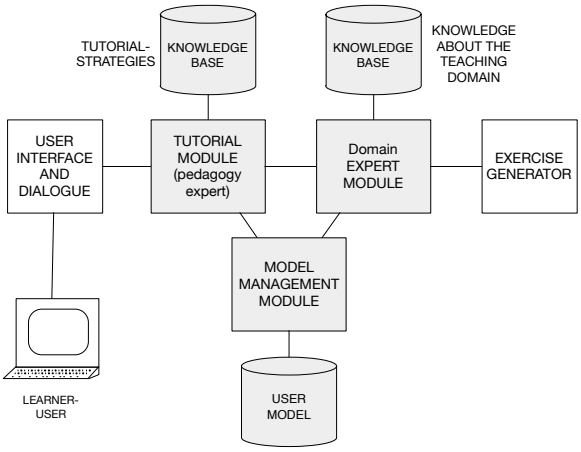


Abbildung R-II2b-9: Architektur eines ITS mit Pädagogikmodul [1999 LELOUCHE]

2000er

Ab der Jahrtausendwende gibt es einen Abfall der Euphorie für den Einsatz von E-Learning zum Einsatz für die automatisierten Lehre. Um diesem Abfall entgegen zu wirken wurde eine Mischform aus E-Learning und dem klassischem Unterricht modern: „Blended learning“ geschaffen. Es ist zwar ein Lehrer anwesend, doch wird der Frontalunterricht durch E-Learningsysteme aufgeweicht, sodass die Lerner zumindest teilweise selbstständig mit dem Computer interagieren müssen. Dies galt natürlich auch für den Einsatz von ITS.

Viele der schon vorhandenen Funktionen und Funktionskonzepte wurden automatisiert oder verbessert. Fehlerdatenbanken und das Anzeigen von Hinweisen gab es schon vorher - beides wurde aber jetzt in ITS integriert und erweitert. In Fehlerdatenbanken werden häufig gemachte Fehler von Lernenden integriert und dann als Strukturen für die Rückmeldung an den Lernenden genutzt. Die Fehlerrückmeldung wird dabei im Laufe der Zeit

immer feiner und fokussierter. Hinweise sind dazu da , den Lerner bei der Bearbeitung der aktuellen Aufgabenstellung zu unterstützen, ohne ihm konkrete Lösungen zu verraten.Besonders die automatisch generierten Fehlerdatenbanken [2008 SUAREZ and SISON] und Hinweise [2008 NIELSEN, WARD and MARTIN] sind ein Schritt in die Richtung, dass die ITS weniger fertiges Material brauchen um intelligent zu agieren, sodass es nicht notwendig ist, für viele Fälle eine von Menschen erstellte Datenbank von fertigen Hinweisen und häufig gemachten Fehlern zu haben.

Durch die Möglichkeit jetzt mit dem Computer Lehrstrategien in einem einfachen Format auszuprobieren kam es zu einigen ITS-Entwicklungen mit außergewöhnlicheren, ursprünglich lerntheoretischen oder didaktisch fundierten Strategien. Zum Beispiel gab es Systeme zum Lehren durch fehlerbehaftete Beispiele [2010 TSOVALTZI, MCLAREN and MELIS]. Ein anderes ist Lehren durch lehren lassen, was die Frage beantworten will, wie gut man lernt wenn man dem ITS versucht etwas beizubringen [2010 MATSUDA, KEISER and RAIZADA]. Oder auch Lehren ohne Text sondern mit Diagrammen [2008 PINKWART, LYNCH and ASHLEY].

Es wurde sich in diesem Jahrzehnt auch damit beschäftigt, wie Lernende motiviert werden können. In den USA war der Umsatz mit Computerspielen 2005 größer als in der Filmindustrie und 2007 größer als die Musikindustrie [2018 WIKIA.COM]. Die Idee basiert auf der Beobachtung, dass Lernende freiwillig stundenlang Computerspiele spielen - nicht jedoch die gleiche Intensität oder Dauer bei der Bewältigung von Bildungsmaterialien auf dem Computer an den Tag legen. Die Idee ist die Motivation der Lerner in dieser interaktiven digitale Unterhaltung für digitale Lehrsysteme zu nutzen. Es entstanden die Begriff „Game-based Learning“ und „Gamification“. Game-based Learning bedeutet dabei das Lernen anhand eines an Spielkonzepten angelehnten Designs des Lernsystems. Gamification ist die Anreicherung einer „normalen“ Lernumgebung mit einzelnen spielerischen Aspekten, beispielsweise Scores und Levels [2008 MARTENS]. Fächer wie Mathematik eignen sich offenbar besonders gut hierfür [2012 RAI and BECK]. Daraufhin gab es Kritik, dass solche Systeme nicht wirklich lehren würden [2005 PRENSKY and BOWERS]. „Serious Games“ sind eine Antwort darauf genau dieses Problem zu lösen, indem sie weniger Spiel und mehr ernsthafte Lehre in ein weniger oberflächlicheres auf Spaß getrimmtes Computerspiel integrieren ohne dabei in eine trockene Simulation abzudriften, die versucht möglichst realitätsgetreu zu sein und dabei Spielspaß und Motivation verliert.

Weitere Entwicklungen war die Idee des Open Learner Models (OLM), beim dem es dem Lernenden ermöglicht wird sein Lernermodell detailliert einzusehen und teilweise auch zu bearbeiten. Es soll mehr Transparenz schaffen und vertraut darauf, dass die Lernenden nicht das System betrügen.

2010er

Die Forschung konzentriert sich in den letzten Jahren darauf, den Lernenden zu analysieren. Es sind einige Forschungsarbeiten vorhanden, wie zum Beispiel [2012 JRAIDI, CHALFOUN and FRASSON] oder [2012 LIN, WANG and CHAO], welche die Emotionen der Lernenden betrachten. Ein ITS kann damit in der Beurteilung des Lernenden mehr einfließen lassen als bloßen faktisch bewertete Antworten. Dies erfolgt anhand von subjektiver Selbsteinschätzung oder in Ausnahmen auch emotionaler Einschätzung durch Gesichtserkennung [2005 SIDNEY, CRAIG and GHOLSON] als auch von der Auswertung von Gehirnwellenanlayse [2018 XU, ZHOU and WANG]. So kann die automatisch Erkennung des Zustands des Lernenden, unter anderem zum Beispiel die Gemütslage und die Konzentration, weitere Möglichkeiten zur Adaption des ITS bringen. Das zweite Gebiet welches heraussticht ist die Bewertung von Lehrmaterial im ITS. Hierzu werden Daten bei dem Lernenden gesammelt um schlechte Lehrmaterialien herauszufiltern und Gute zu analysieren, warum diese gut sind. Es werden in diesem Jahrzehnt immer mehr ITS-Prototypen in Gebieten verwendet, welche nicht mit Regeln zu erfassen sind. Anstatt Ma-

thematik und Physik werden auch Sprachen, Argumentation oder die Erkennung von Täuschung implementiert. Ein ITS welches zum Beispiel versucht eine politische Diskussion zu führen, anstatt rein zu lehren.

Die Komplexität ist von ITSs so groß geworden, dass immer mehr Forschungsthemen sich nur noch auf eine Komponente konzentrieren. Auch wenn ITS Architekturen trotzdem bis zu diesem Entwicklungspunkt selten thematisiert wurden.

Die starke Veränderung der digitalen Lehrsysteme durch die explosionsartige Verwendung von Smartphones und Tablets in der Lehre und deren hohe Pixelanzahl und Bildschirmgröße mit zugleich nicht zu verachtender Leistung sowie ihrer Mobilität hat auch dazu geführt, dass Lehrsysteme (besonders Lehrprogramme die verkauft werden) auf diese neuen Endgeräte setzen. Leider führte das nicht zu einer Entwicklung der ITS in seiner Kernfunktionalität sondern nur zu einer Portierung oder Erweiterungen der restlichen Funktionen. Als Beispiel sei hier die Kollaborationsfähigkeit genannt, bei der es Lernenden möglich ist rein digital zusammen zu arbeiten und damit dem ITS mehr Chancen der Auswertung von Daten geben. Es kommen immer weitere Softwarearchitekturen und ITS-Implementierungen hinzu. Deren Evaluation an einem Lehrmodell ist interessant und relevant. Das Grundmodell eines ITS hat sich nicht entwickelt. Die Architekturen beinhalten sogar häufig nicht mehr die Pädagogikkomponente, siehe dazu Abbildung R-II2b-10.

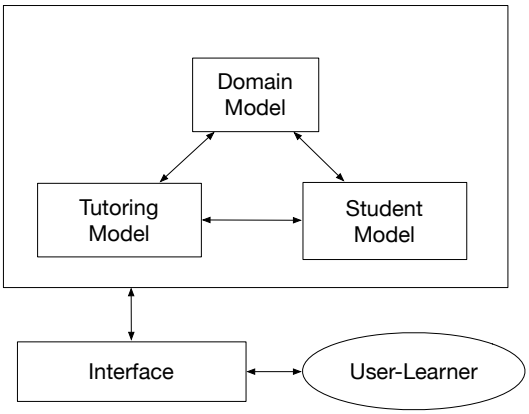


Abbildung R-II2b-10: 4 Komponenten Modell [2010 NKAMBOU, MIZOGUCHI and BOURDEAU]

II.2.c. Lehrsysteme in der Konsumwirtschaft

Analyse des Istzustands für ein E-Learningsystem als Produkt unabhängig von Institutionen. Wie weit lassen sich die Ideen in der Forschung Marktwirtschaftlich umsetzen und sind gleichzeitig im großen Stil erfolgreich. So wird kurz analysiert welche erfolgreiche Lehrsoftware aktuell ist. Es gibt verschiedene Kategorien für die Spiele und Lernprogramme. Der Lehrgedanke ist mehr oder minder stark vertreten. Exploratives Lernen und dahingehend auch Simulationen sind besonders gut angenommen, weil sie es dem Lerner explorativ erlauben sich mit der Materie zu beschäftigen [1991 JONG] und dabei muss das didaktische Modell nicht so stark ausgeprägt sein. Mit „game based learning“ wurde der Trend zu Computerspielen auch aufgrund von leistungsstärkeren Computern aufgenommen und mit seriösem Lernen im „serious games“ verbunden.

Erfolgreiche Lehrsysteme

- Physikpuzzles (zum Beispiel „Crazy Machines 3“ [2018 FAKT SOFTWARE and DAEDALIC ENTERTAINMENT])
- Raumfahrt und Physikbaukästen (zum Beispiel „Kerbal Space Program“ / „Kerbal-EDU“ [2018 SQUAD and TEACHERGAMING])
- Ökonomiesimulationen (zum Beispiel „Cities Skylines“ [2018 COLOSSAL ORDER LTD. and PARADOX INTERACTIVE])
- Flugsimulatoren (zum Beispiel „X Plane“ [2018 LAMINAR RESEARCH])
- Programmieren lernen (zum Beispiel „Robot School“ [2018 NEXT IS GREAT SP Z O.O.] oder „WeHeartSwift Exercise Platform“ [2018 WE ❤ SWIFT])
- Mathematiklernprogramme (zum Beispiel „DragonBox Algebra 12+“ [2018 WE-WANTTOKNOW AS])
- Single-Choice Prüfungslernprogramme (zum Beispiel „iTheorie“ [2018 SWIFT MANAGEMENT AG] oder „AMBOSS“ [2017 AMBOSS])

Um eine Privatperson dazu zu kriegen ein Lehrsystem mit nicht zu unterschätzenden Preisen von 20 bis 60 Euro eigenständig zu bezahlen, erfordert es meistens einen Spielaspekt, abgesehen von den Prüfungslernprogrammen, welche speziell auf eine ganz bestimmte Prüfung entwickelt wurden. Diese auf eine Prüfung mit einfachen Antwortmuster zugeschnittenen Lehrsysteme brauchen als Voraussetzung eine Prüfung die sich nur wenig und in der Struktur in Jahrzehnten kaum bis gar nicht geändert haben mit vielen Lernenden. Besonders rentabel sind dabei Lehrprogramme für Examensprüfungen aller Art, von Jura, über Medizin und auch staatliche Erlaubnisprüfungen wie die verschiedensten Fahrerlaubnisprüfungen von Boot bis LWK. Meistens sind solche Prüfungen Single-Choice-Tests. Deren Komplexität ist durch die geringe Antwortmöglichkeiten deutlich geringer als ein Lehrsystem mit freien Antwortfeldern. Als Beispiel gibt es im Medizinstudium zwei schriftliche Examen, welche jeweils auf einem Single-Choice-Test mit immer genau 5 Antworten beruhen. Diese sind prädestiniert dazu in einem einfachen Lehrsystem verwendet zu werden, weil die Eingabemöglichkeiten des Benutzers vorerst auf 5 Antworten reduziert sind. Erweiterungen sind denkbar, allerdings wird höchstens die die Auswahl keine Antwort zu geben hinzugefügt, komplexe Systeme mit zum Beispiel einem Einbeziehen von Tags von Kommentaren sind bisher nicht erfolgreich verkauft worden. All die verschiedenen Lehrsysteme für Medizin sind sich ähnlich. Eines sticht durch seine Benutzerfreundlichkeit und großer Funktionalität heraus: Amboss [2017 AMBOSS]. Die Hauptkosten entstehen dabei durch die Lizenzierung der Fragen vom Prüfungsamt für Gesundheit und Soziales und den Gehältern von Ärzten welche Kommentare für die jeweiligen 6 Antwortmöglichkeiten (5 Antworten oder nichts beantwortet zu haben) erstellen. Die Entwicklungskosten für eine Webseite oder App sind überschaubar mit unter 20'000 Euro zu veranschlagen.

Es wäre wünschenswert, wenn Firmen mehr in ein ausgeklügelteres Lehrsystem investieren würden. Die Aussage der Firmen ist meist: Es kauft ja niemand. Die Aussage der Lernenden ist, es ist nicht gut genug ausgereift. Wir sind also in einer Hennen-Ei-Situation bei der irgendwann eine Firma das Risiko eingeht um ein Lehrsystem zu entwickeln, welches teuerer, aber nicht absurd teuer und umfangreicher als die bisherigen Systeme ist und damit genügend erfolgreich ist, dass andere Firmen es ihr gleichtun wird um nicht ins Hintertreffen zu geraten. Ähnliches passierte mit Smart Home Entwicklungen in den letzten 5 Jahren.

Einsatzbereiche im größeren Stil

Da eine Entwicklung eines ITS so schwierig ist bei Privatpersonen zu verkaufen oder einzusetzen, ist es nicht überraschend dass die angepeilten Einsatzbereiche von ITS in Lehrinstitutionen liegen. ITS können im Gegensatz zu menschlichen Lehrpersonen vielfach gleichzeitig und ohne Pause eingesetzt werden. Der Vorteil macht sich besonders in größeren Einsatz wirklich bemerkbar. Es gibt schon viele Bereiche welche Lehrsysteme im großen Stil verwenden und durch Lizenzgebühren bezahlen, die Bereiche sind folgend aufgeführt:

- Schulen
- Universitäten
- Selbststudium
- Weiterbildungen für die Wirtschaft
- Militär

Hier könnte ein ITS vorherrschenden Lehrsysteme ablösen, aber nur wenn das eingesetzte ITS besser ist. Da es in den meisten Einrichtungen schon Lehrsysteme gibt muss für einen Wechsel die Differenz groß genug sein, dass sich der Aufwand eines Wechsels ausgleicht.

II.2.d. Was ist ein Intelligent Tutoring System?

Intelligente Tutoring Systeme, oder auch in deutschen Sprachgebrauch intelligente Lehr-/Lernsysteme sind im Bereich zwischen Künstlicher Intelligenz der Informatik, Kognitionspsychologie und Didaktik/Training angesiedelt, siehe hierzu ein Mengendiagramm in Abbildung R-II2d-1. Deren Abkürzung sind ITS oder ILLS, der weitaus häufiger genutzte Begriff ist die englische Variante, deshalb wird fortan ITS als Abkürzung in dieser Arbeit verwendet. In [1990 NWANA] gibt es das Mengendiagramm auch, dort fehlt allerdings die Benennung der, die Benennung des ITS-Bereichs in des in der Mitte wird zusätzlich explizit mit "Cognitive Science" beschrieben. Die ITS basieren auf der kognitiven Psychologie als die zugrundeliegende Lerntheorie, welche hauptsächlich mit den Problemen der Organisation der Wissensrepräsentation des menschlichen Gehirns beschäftigt ist. Ein ITS kann, aus heutiger Sicht, im Gegensatz zu anderen Lehrprogrammen komplexe Antworten vom Lerner interpretieren und vom Verhalten des Lernenden lernen was dieser weiß und wie gut seine Fertigkeiten sind. Es ist damit dem ITS möglich Fragen zu stellen, welche passend zum Lernenden sind um nicht nur mit richtig und falsch die Antworten des Lernenden zu bewerten. ITSs sind daher ein Forschungsgebiet, dass eine fächerübergreifende Disziplin darstellt, bei dem sowohl das technische Verständnis eines komplexen Computersystems, sowie Fachwissen über Lehrmaterial, pädagogische Wissen für die Regeln von Verhaltensweisen und didaktisches Wissen zum Zwecke der Entscheidung was wie unterrichtet wird benötigt werden. So überrascht es nicht, dass es unterschiedliche Sichten auf ITSs gibt, welche die Erstellung von eben jenen beeinflusst. Der unterschiedliche Einfluss von den Gruppen und deren verschiedene Ansichten lassen auch ungleiche ITSs entstehen.

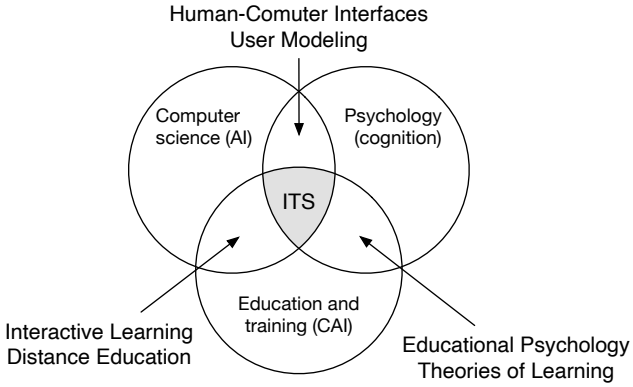


Abbildung R-II2d-1: Domänen eines ITS [2010 WOOLF]

Ein Kategorisierung von ITSs ist, dass sie in ihrer Fähigkeit unterschieden werden, welche diese mit auszeichnen: Deren Adaptivität. Welche Komponenten adaptiv sind hat große Auswirkungen in einem ITS. In ITSs wird häufig der Dialog als Interaktionsmittel verwendet, weil man sowohl Feedback geben möchte als auch Informationen von dem Lernenden braucht, die flexibler sein müssen als eine Auswahl von einer kleinen festen Anzahl an Möglichkeiten. Im Gegensatz zu einer virtuellen Welt sind die Kosten der Implementation einer zweidimensionalen Benutzeroberfläche einfach, überall auffindbar, benötigen wenig Leistung und erlauben einen hohen Detailgrad und eine hohe Genauigkeit der Eingaben. Zudem sind Analysen von Texten in der künstlichen Intelligenz viel erforscht, dafür stehen schon Softwarelösungen bereit. Andere Interaktionsmöglichkeiten wurde weiter oben schon besprochen.

Einer der ersten Definitionen von einem ITS ist laut [1982 SLEEMAN and BROWN], dass es ein Computerlernprogramm ist, welches künstliche Intelligenz nutzt um Wissen zu repräsentieren und eine Interaktion mit dem Nutzer aufrecht zu erhalten. Der Nutzer ist der Lernende. Im Jahr 2000 wurde ein ITS von Freedman et al [2000a FREEDMAN, ALI and MCROY] folgendermaßen definiert:

> "intelligent tutoring system" is a broad term, encompassing any computer program that contains some intelligence and can be used in learning.

Eine Konkretisierung des im Zitat verwendeten Intelligenzbegriffes nehmen Paroli und Kairam vor [2013 PIROLLI and KAIRAM]:

> [...] a core element of what makes an Intelligent Tutoring System (ITS) "intelligent" is its ability to accurately diagnose students' knowledge and adapt instruction accordingly.

Hieraus wird eine folgend erwähnte allgemeine Definition ableitet.

- Ein ITS ist ein Softwaresystem zum Lehren von Domänenwissen. Dabei wertet das System die langfristig gespeichertes Nutzerverhalten aus und gibt eine zum Lernenden angepasste Rückmeldungen.
- An ITS is a software system to teach expert knowledge. The system has to evaluate the long term collected user behavior and give a for the learner customized feedback.

Von Hartley und Sleeman [1973 HARTLEY and SLEEMAN] stammt einer der ersten Anforderungslisten für ein ITS. Folgend aufgelistet sind die drei Anforderungen um die Lehraufgaben zu bewältigen drei weitere um die Lerneffizienz zu verwalten.

- ITSs sollten in der Lage sein ihr eigenes Übungsmaterial zu erzeugen / „the programs should be able to generate or select material to compose a learning task“
- ITSs sollten in der Lage sein ihr Übungsmaterial schrittweise zu lösen / „they should be able to generate intermediate and final solution states“

- ITSs sollten in der Lage sein bestimmte Fehler und Fehlinterpretationen durch Abgleich mit ihren eignen Lösungen des Lernenden erkennen können / „they should be able to recognize types of errors or misconceptions through the effects they have on the solution states“
- ITSs sollten in der Lage sein alternatives Lehrmöglichkeiten für die gleichen Lehrziele/gleiches Domänenwissen bereit zu stellen / „the programs should be able to provide alternative teaching operations for attaining the same educational objectives
- ITSs sollten ihr Verhalten an die Leistung des einzelnen Lernenden anpassen / „the programs should have means ends guidance rules to match teaching operations to individual performances
- ITSs sollten in der Lage sein sich selbst durch mehr Daten zu verbessern / „the programs should be capable of improving with experience.

KI ist damit eines der zentralen Fähigkeiten eines ITS um diese Anforderungen zu erfüllen. Die KI in ITS innerhalb der Forschung dient vor allem dem Erforschen von KI beim Lehreinsatz. Die KI verbessert die Interaktion des ITS mit dem Lernenden, indem das Lehrmaterial und die Rückmeldungen an den Lernenden angepasst werden. Es dient zur Erhöhung der Motivation, das zu vermittelnde Wissen besser zu analysieren und einer effizienten Wissensvermittlung durch Adaption des Programms an den Lernenden. Dies soll einen persönlicheren Tutor erzeugen, anstatt nur ein digitalisiertes Lehrbuch. Es schätzt das Wissen des Lernenden ein und was seine Präferenzen und Schwächen sind und kann sich mit dem Lernenden weiterentwickeln.

Eine der Schlüsselverhaltensweisen von ITSs ist, dass auf die Bedürfnisse der Lernenden eingegangen werden kann. Dies geschieht in einem ITS durch das automatische Personalisieren an den Lernenden während der Benutzung des Systems. Die Adaptivität ist die Anpassungsfähigkeit des ITS an jeweils den einzelnen Lernenden. Die Modellierung der Abbruchquoten bei der Lehre mit einem ITS zeigt wie wichtig personalisierte Antworten für die Motivation sind [2014 EAGLE and BARNES].

Eine Möglichkeit der Anpassung ist es die Antworten des ITS so zu wählen, dass sie bestmöglich den Lernenden zum Lehrziel leitet. Die Antworten können durch eine Kombination von konkreten vorgefertigten Antworten, vorhandenen Lehrmaterialen und komplett automatisch adhoc generierte Antworten entstehen. Meist werden die Antworten in Textform generiert, es ist heutzutage möglich diese mit einer synthetischen Stimme auszusprechen. So entsteht ein Dialog zwischen dem Lernenden und dem ITS und die Adaptivität ist durch die Anpassung der Ausgabe des ITS an die Eingabe des Lernenden gegeben. Einer der großen Vorteile eines ITS ist seine Kopierbarkeit. Im Gegensatz zu einem Lehrer, sinkt die Adaptivität nicht mit zunehmender Anzahl an Lernenden, weil mehr als eine Ausprägung eines ITS Gleichzeitig existieren kann. Um der Aufgabe der Anpassung gerecht zu werden wertet das ITS die Eingaben des Lernenden aus, so kann es den Lernenden einschätzen. Es wird nicht genau eine festgeschriebene Handlung erwartet, sondern auf einen größeren Rahmen von Handlungsmöglichkeiten reagiert. Dabei muss die Eingabe nicht immer bewusst erfolgen, es können auch die Aufmerksamkeit und das Befinden des Lernenden selbst eine unbewusste Eingabe sein. Bei Eingaben zum Lehrstoff wird nicht nur die Korrektheit zur eigentlichen Aufgabenstellung, sondern es wird auch versucht den Gedankengang zu erkennen. Bei fehlerhaften Eingaben ist es sehr nützlich einzuschätzen, warum der Lernende diesen Fehler gemacht hat. Ob der Lernende unaufmerksam war, es an Domänenwissen fehlte oder ist etwas missverstanden worden. Bei korrekten Eingaben können auch Rückschlüsse auf beim Lernenden vorhandenen Basiswissen geschlossen werden.

Es gibt natürlich auch andere Methoden adaptiv zu sein. Um ohne Eingabe adaptiv zu sein muss vorerst die Fertigkeit des Lernenden in den für die Interaktion mit dem ITS relevanten Themen analysiert werden. Mit einer naiven Annahme, dass der Lernende alles verstanden hat, was ihm angezeigt wurde, sind die angezeigten Informationen gleich der Menge an Informationen, welche von dem Lernenden verstanden wurde und damit alle Aufgaben welche diese darauf aufbauen theoretisch lösbar. Schwieriger zu implementieren ist es dass das Verständnis von einem Lehrstoff in mehreren Ebenen vorhanden sein kann. Egal welche Methode man wählt, eine der Anpassungen kann der Schwierigkeitsgrad der Lehrsitzung sein. Der Schwierigkeitsgrad kann mannigfaltig variiert werden, zum Beispiel indem weniger Hilfsinformationen angezeigt werden, ihm ein Zeitlimit zum Lösen der Aufgabe zu setzen oder von vorn herein eine komplexere Aufgabe zu stellen. Es gibt also beim ITS adaptives Verhalten, welches nicht direkt auf einem Aktions-Reaktions-Schema mit kurzen Zeitabständen beruht, sondern ein allgemeines Anpassen des Systems, solange es zu gesteigerter Lerngeschwindigkeit führt.

Zusammenfassend ist ein ITS...

- ...ein E-Learningsystem
- ... ein System welches künstliche Intelligenz verwendet
- ...adaptiv zum Lernenden (seine Lerngeschwindigkeit und Stand der Expertise)
- ...ein System welches dem Lernenden eine Rückmeldung zu seinen Aktionen gibt, die erklärt und/oder unterstützt

II.2.e. ITSs in der Forschung

Die folgende Auflistung, siehe Tabelle R-II2e-1, der ITSs zeigen die in der Literatur aufkommenden und wichtigsten Systeme und bilden die Grundlage für heutige ITSs. Eine geschichtlichen Überblick über konkrete ITSs deren Entwicklungen und Beziehungen untereinander gibt es als Forschungsarbeiten es schon sehr ausführlich [1999 LELOUCHE] [2004 MARTENS].

Name	Jahr	Name	Jahr	Name	Jahr	Name	Jahr
Automatic Teacher	1930	WEST	1978	HERACLES	1987	Mainstream Touchscreen Smartphones with iPhone	2007
Erster Programmierbarer Computer: Simon	1949	ACT	1980	MAIS	1987	eTeacher	2008
Exploratories	1960	Notecards	1980	PIXIE	1987	Realise-IT	2009
FRESS	1960	Perseus	1980	SIERRA	1987	ZOSMAT	2009
HES	1960	IBM-PC	1981	Talus	1987	The Cognitive Tutor (Cognitive tutors)	2010
IBM 1500 PLATO	1960	LMS	1981	SCENT-3 Advisor	1988	Wayang Outpost	2010
Augment/NLS	1962	NEOMYCIN	1981	Kommerzialisierung des Internets und das WWW	1989	Mainstream tablet with iPad	2010
BASIC	1964	First mass produced laptop: Osborne 1	1981	ABSYNT	1990	ASSISTments	2012
DENDRAL	1965	Calculus Tutor	1982	Atlas Andes	1990	calculus@internet	
Erster PC HP-9100A	1968	DEBUGGY	1982	BALSA	1990	CASTER	
META-DENDRAL	1970	EXCHECK	1982	Boxer	1990	CODES	
SCHOLAR	1970	GLisp	1982	D3 Trainer	1990	Crystal Islands	
MYCIN	1972	INTEGRATION Tutor	1982	Educational Fusion	1990	DEBUG	
METEOROLOGY ITS	1973	MACSYMA Advisor	1982	ELM-ART	1990	HyperCard	
AM	1974	Model of Endorsement	1982	FITS	1990	ILEX	
EMYCIN	1974	QUADRATIC Tutor	1982	MEDICUS	1990	Intermedia	
TEIRESIAS	1974	SOPHIE	1982	MOOSE Crossing	1990	Knewton	
ACE/PSM	1975	WHY	1982	PETRI-Help	1990	LiveWorld	
AI Handbook	1975	WUSOR	1982	SHERLOCK	1990	LOGO	
MALT	1975	ARITHMEKIT	1983	STEVE	1990	MANAGE	
MOLGEN	1975	ATDSE	1983	Virtual Laboratory at University of Oregon	1990	Mathematics Tutor	
UNITS	1975	EDSMB	1984	PAT	1993	MECC	
AGE	1976	LISP Tutor	1984	CIRCSIM Tutor	1994	MethodMan	
BIP	1976	MENO-Tutor	1984	ISIS-Tutor	1994	Object Logo	
Contract Nets	1976	PROUST	1984	ANDES	1996	OGF	
CRYSALIS	1976	STEAMER	1984	C++ Tutor	1996	PACT	
SPADE	1976	Erste GUI: Macintosh I	1984	Cardiac Tutor	1996	Project LISTEN	

Name	Jahr	Name	Jahr	Name	Jahr	Name	Jahr
CENTAUR	1977	ALGEBRALAND	1985	Erstes Mainstream Smartphone Nokia Communicator	1996	REALP	
FLOW Tutor	1977	Apple Classrooms of Tommorrow	1985	C-Tutor	1997	SACON	
GUIDON	1977	BRIDGE	1985	ADELE	2000	SmartTutor	
PUFF	1977	GEOMETRY Tutor	1985	Summary Street	2001	Stanford CAI	
SACON	1977	QUEST	1985	SQL-Tutor	2002	StorySpace	
VM	1977	SPIRIT	1985	Why2-Atlas	2002	The Geographer's Craft	
Commodore PET 2001	1977	THEVENIN	1985	InterMediActor	2003	ThingLab	
BLOCKS	1978	TUTOR	1985	Web 2.0 und die Cloud	2003	TICCIT	
BUGGY	1978	FGA	1986	ActiveMath	2004	UC Irvine CAI	
EURISKO	1978	EUROHELP	1987	VisMod	2004	VIVARIUM	
QUIST	1978	GERMAN Tutor	1987	AutoTutor	2005	WATCH	
RLL	1978	GUIDON2	1987	ASPIRE	2006	WE	

Tabelle R-II2e-1: Zeitlinie der ITS

Manchmal ist es nicht möglich genau herauszufinden in welchem Jahr ein ITS eingesetzt worden ist. Der zeitliche Verlauf in der Tabelle zeigt, dass die Entwicklung der ITSs mit der Entwicklung der Computer zusammenhängt. Wie schon in der der Historie der Lehrsysteme in einem vorherigen Kapitel erklärt entstanden Lehrsysteme als eigenständig reagierenden System durch den „Automatic Teacher“ aber erst durch den Computer war es möglich solche Systeme komplexer, günstiger und kopierbar auf andere Computer zu entwickeln, die CAI-Systeme. Viele CAI-Systeme wurden in der Forschung entwickelt. CAI-Systeme stellen Vorläufer der ITSs dar, in dem Sinne, dass beide Systemtypen das Ziel der automatisierten Lehre haben. Erst als Computer schneller wurden und komplexere Berechnungen in kürzerer Zeit erlaubten, war das Konzept eines adaptives automatisierten Lehrers (wie ein ITS) durch künstliche Intelligenz realisierbar. Auswertungen der natürlichen Sprache um Dialoge mit einem Computer in einer Domäne zu führen waren Anfang schon der 80er ausgereift genug [1983 O'SHEA and SELF], was zu einem Hype in ITS führte. Dieser Hype nahm um das Jahr 2000 dann wieder stark ab, siehe Abbildung R-II2e-2. Weniger intelligente, sondern eher Verwaltungs- und Softwarewerkzeuge die sich an der Präsentation von Lehrmaterial orientierten, waren die de facto praktisch eingesetzte Software in der Lehre und auch das hatte Auswirkungen auf die Forschung.

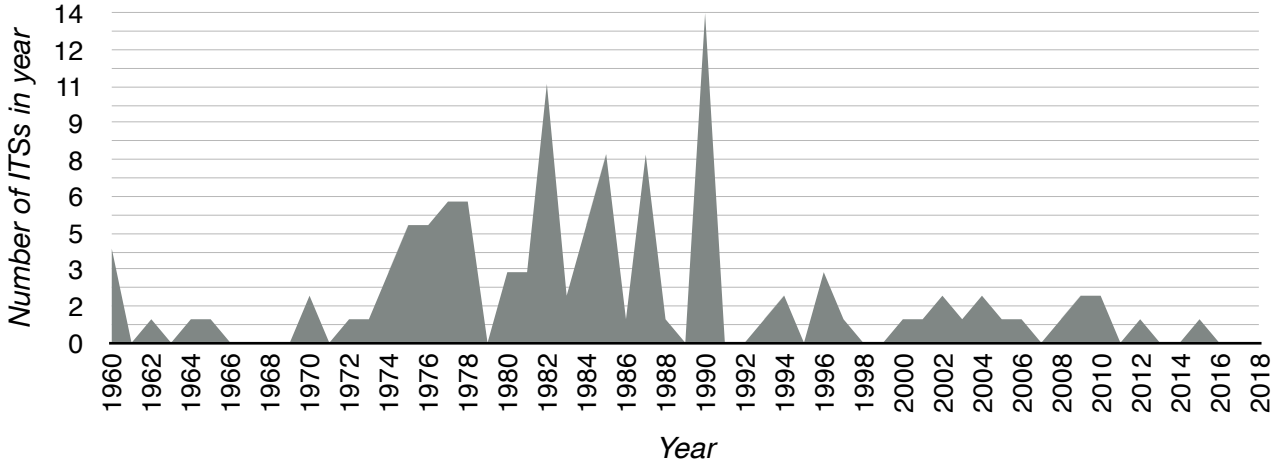


Abbildung R-II2e-2: Anzahl der gefundenen ITSs pro Jahr [2019b GRAF VON MALOTKY and MARTENS]

ITSs sind außergewöhnlich in ihren Methoden und fundamentalen Theorien. Erst 1970 hat Carbonell einen Prototypen SCHOLAR entwickelt, welcher den Begriff des ITS prägte und den ITSs das „I“ für „Intelligent“ gab. SCHOLAR war ein Programm zum Lernen von Erdkunde. Dieses ITS interagierte mit dem „Sokratischem Dialog“ mit dem Lernenden. Das Wissen wurde mittels semantischer Netze gespeichert und damit auch die Verbindungen zwischen den Einzelnen Informationen. So könnte man sagen, dass die ITSs dort ihre Geburt hatten, doch wie bei so vielen Dingen ist die Entwicklung nicht sprunghaft entstanden. Der Ursprung dieser Systeme ist klar und kommt von den Computer Assisted Instruction (CAI). Es gab vorher das CAI und die Tutoring Systeme, CAI-Systeme wurden dann umbenannt zu ICAI und schlussendlich zu ITS. Somit sind ICAI-Systeme Synonyme für ITS, der Begriff ICAI wurde durch ITS verdrängt, wahrscheinlich um klar zu machen, dass ITS weitaus mehr bedeutet als eine kleinere Weiterentwicklung eines CAI-Systems.

Eines der größten CAI-Systeme ist PLATO, es war nicht nur ein Programm, sondern ein ganzes Lehrsystem mit kompletter Hardware. In PLATO wurden später auch andere ITSs als neuere Programme für das System entworfen, eines der bekanntesten ist WEST. WEST ist ein Brettspiel auf dem Computer, bei dem zum Bewegen der Figur Zahlen mit mathematischen Operatoren kombiniert werden müssen um zu gewinnen. Das ITS gibt Verbesserungsvorschläge für Spielzüge lehrt implizit dadurch Mathematik.

Es konnte Lehrmaterial in einem englischen Dialog erzeugen, denn es konnte unbekannte Fragen beantworten durch ableiten von Schlüsselwörtern und schlussfolgern von Attributen auf Wissensbausteine. Das Domänenwissen waren die geographischen Informationen über Südamerika. Das ITS versuchte also ein Gespräch mit einem Lernenden aufzubauen, welches dem eines menschlichen Lehrers ähnelt. Die Architektur entspricht der klassischen Architektur.

SOPHIE ist auch ein bekanntes ITS in der Forschung, welches nicht versucht einen menschlichen Lehrer zu imitieren, sondern ein Lernwerkzeug zu schaffen in dem zum richtigen Zeitpunkt Ratschläge und passende Wissensbrocken für die aktuelle Handlung bereit gestellt werden. Die Umgebung vermittelte Wissen über Elektronik und wie man Fehler in elektronischen Schaltkreisen beheben kann. Der Lernende konnte auch aktiv Wissen nachfragen. Mit einer Entwicklungszeit von 5 Jahren war es das erste vollständige ITS als ein Produkt, doch ein großflächiger Einsatz außerhalb der Forschung blieb aus.

Der Erfinder der ersten ITS-Softwarearchitektur (William J. Clancey) entwickelte später ein ITS mit dem Namen GUIDON. Es war der Versuch ein sehr bekanntes Expertensystem mit dem Namen MYCIN in ein ITS umzuwandeln. MYCIN ist ein medizinisches Expertensystem, dass Behandlungsweisen für Bakterieninfektionen vorschlägt. GUIDON vergleicht die Antwort eines Lernenden mit der Antwort des Expertensystems, wobei die Lösung

des Expertensystems als korrekt angesehen wird. Ein ITS ist damit nur ein Zusatz zu einem Expertensystem mit Lehrstrategien und einem Lernermodell. Später wurde MYCIN weiterentwickelt zu NEOMYCIN und darauf aufbauend wurde auch GUIDON2 weiterentwickelt.Es konnten immer wurde kleinere Lektionen zum Lernen zur Verfügung gestellt und vom ITS unterstützt [1986 CLANCEY], was allerdings wenig Auswirkungen auf die allgemeine Softwarearchitektur hatte.

Es gab unter den Systemen auch welche die den Konstruktivismus vertraten. Einer der bekanntesten Vertreter ist die 1980 in einem Buch veröffentlichte Lernumgebung LOGO mit der gleichnamigen Programmiersprache. Sie ließ dem Lernenden freie Hand beim Lernen mit explorativen Welten ohne Anleitung [1980 PAPERT].

Über die Internetsuche wurden über trotzdem immer wieder einige kurze ITS-Beschreibungen gefunden, zu welchen es kaum Informationen gab, zum Beispiel IKITS als ein ITS für Chinesisch [2018 KOKKINAKIS]. Die Dunkelziffer an der Anzahl an Entwicklungen von ITSs muss noch höher sein als in der obigen Liste erkennbar.

II.2.f. Nachteile eines ITS im Allgemeinen

Die Kosten für die Entwicklung eines ITS sind bisher sehr hoch. Die Entwicklungszeit bis zur Fertigstellung kann mehrere Jahre betragen. Die Kosten für die Wartung des Systems und die Verwaltung darf nicht vernachlässigt werden [2008 ANDERSON]. Diese Ressourcen für E-Learningsysteme könnten auch in Werkzeuge für Autoren und einfacheren Lehrsystemen als ITS investiert werden, um es den Autoren von Lehrmaterialien zu erleichtern besseres digitales Lehrmaterial zu erzeugen. Ein intuitives Autorenwerkzeug macht sich durch seine geringere Komplexität schneller positiv in der Lehre bemerkbar als ein ITS, alleine schon durch die kürzere Entwicklungszeit. Wenn man die Ressourcen eines ITS zur Verfügung hat könnte auch kein ITS, sondern weniger komplexes E-Learningprogramm mit weniger Kosten entwickelt werden. Zusätzliche Ressourcen könnten in eine bessere Benutzeroberfläche und Benutzerinteraktion investiert werden. So kann ein funktionell schwächeres E-Learningprogramm bei gleichen Kosten wertvoller wirken oder sein als ein ITS. Die Frage stellt sich ob sich ein ITS für eine womöglich erhöhte Lehreffizienz im Vergleich zu statischen Selbstlernmaterialien lohnt. Die entstehenden Kosten für ein ITS fallen gleich in mehrere Bereiche: Es muss softwaretechnisch, didaktisch als auch vom Domänenwissen auf einem gewissen Entwicklungsstand sein, um einsetzbar zu sein. Studien legen nahe, dass ITSs zwar exzellente Ergebnisse im Wissenszuwachs erreicht, jedoch erfahrene menschliche Tutoren die effektivste Form der bekannten Lehre ist [2006 RAZZAQ and HEFFERNAN].

Die heutigen Ansprüche an ein ITS liegen in der Entwicklung einer künstlichen Intelligenz welche in zwei Bereichen besser werden muss. Zum einen muss die Arbeit für die Erstellung von Lehrmaterial verringert werden. Dies kann durch die automatische Generierung von Lehrmaterial erfolgen. Zweitens muss die Rückmeldung verbessert werden. Für die automatische Generierung von Lehrmaterial müssen die Expertenregeln für die Domäne formalisiert werden, in einer Sprache die das ITS versteht. Da viele Domänen ihre eigene Formalisierungssprache haben die häufig sogar nicht formal genug sind für ein Computerprogramm, kann von dem ITS nicht erwartet werden die vorhandenen Formalisierungen zu verstehen. Von daher muss von dem ITS Autorensystem eine Möglichkeit geschaffen werden. Zudem gibt es für einige Domänen nur wenig Formalisierungen, es wäre denkbar einfache Regeln selbstständig anhand von Beispielen ableiten zu können. Zu den Problemen welche sich Forscher der Künstlichen Intelligenz für ITSs in der Feedbackfunktionalität gegenüber gestellt sieht kann dies zwei geteilt werden: Von einem Lernenden kann man im Gegensatz zu einem ITS nicht erwarten seine Antworten zu sehr an ein ITS anzupassen, daher muss ein ITS sehr flexibel mit dem Lernenden kommunizieren können. Ein ITS muss einerseits verstehen, was der Lernende mit einer zum Beispiel natürlichsprach-

lichen Texteingabe vermitteln will und zu dieser Antwort als auch passend zum Lernenden ein Feedback geben. Die Schwierigkeiten sind dabei teilweise immens, das verstehen von natürlicher Sprache von einer Software ist ein eigener Forschungsbereich.

Die Fortschritte welche oben genannt wurden, werden in der künstlichen Intelligenz und der Anpassungsfähigkeit eines Systems in der heutigen produktiv verwendeten Assistenzsystemen durch ein Mittel umgesetzt welches ein zweischneidiges Schwert sein kann: Eine große Datenbasis über jeden Nutzer sammeln. Ein ITS als ein System welches per Definition adaptiv ist, muss über den Lernenden Bescheid wissen. Wenige Daten reichen da nicht aus, sondern es sind viele Daten die gesammelt werden müssen um gute Ergebnisse zu liefern. Diese Daten können auch missbraucht werden. Denn desto mehr Daten ein System sammelt, umso leichter fällt es diese Person zu identifizieren. Die Privatsphäre muss geschützt und Manipulation vermieden werden.

Um der Komplexität eines ITS Herr zu werden ohne dabei die Qualität des Systems zu verschlechtern, werden sie auf eine Domäne spezialisiert entwickelt. Der Aufwand lässt sich besser abschätzen wenn man sich auf nur einen kleineren Anwendungsfall festlegt und andere ignoriert. Ein konkretes ITS ist damit im Allgemeinen zu spezialisiert um in einer anderen Domäne wiederverwendet zu werden.

ITS sind für einen gewissen Teil der Lernenden eine gute Alternative zu anderen Lernmethoden, allerdings nicht für alle. Die Lernmethode mit einem ITS zu lernen ist nicht inhärent besser als andere, sondern bietet für bestimmte Inhalte, Lernstile und Lernendentypen Vorteile, die einen sinnvollen Einsatz erlauben. ITS sind Programme und Programme sind Werkzeuge. Jedes Werkzeug hat seine Grenzen. Man muss wissen wo die Grenzen des Werkzeugs sind, um es an der richtigen Stelle in der Lehre einzusetzen. Auch wenn die Anpassungsfähigkeit außergewöhnlich hoch ist für eine Software, ist sie im Vergleich zu einem Menschen lächerlich gering. Desto mehr unvorhergesehene Dinge in einem Unterricht passieren, umso schlechter wird die Software sein. In einigen Domänen ist es durchaus gewollt, dass der Unterricht jedesmal unterschiedlich ist weil es zum Beispiel von der Kreativität und den Impulsen der gesamten Gruppe geleitet wird. In diese Formen lassen sich ITS nur schwer integrieren.

II.2.g. Grundkomponenten eines ITS

Laut dem aktuellen Forschungsstand ist die klassische ITS-Softwarearchitektur in ITS vertreten, auch wenn die vorkommenden Komponenten jeweils nicht immer genau in einer Komponente realisiert wurden, siehe dazu die referenzierten Tabellen R-II2g-1 und R-II2g-2 aus den jeweiligen Quellen. In den Tabellen sind jeweils die in den Originalbeschreibungen aufgeführten Namen verwendet. Die Einteilung entsprechend der Komponentenarchitektur erfolgte im Rahmen der vorliegenden Dissertation entsprechend der recherchierten Funktionalitäten. Folgend werden die klassischen Komponenten einzeln erklärt und zwar so wie sie historisch verwendet wurden. Dabei werden die Fachbegriffe für die Komponenten genommen, die besonders häufig verwendet wurden. Folgend werden die Komponenten, deren meistgenutzte Bezeichnung und Funktionsweise beschrieben.

ITS-Name	Benutzerschnitt-stelle	Pädagogikkomponente	Lernermodell	Domänenwissen
LTSA-Standard	Learner Entity, Deli-very	Evaluation, Coach	Evaluation	Coach
A plug-able web-ba-sed Intelligent Tutoring System	User Interface	Tutor Module, Student Module	Student Module	Tutor Module, Expert Module
Intelligent Tutoring System for teaching 1st year engineering	User Interface	Expert System	Students Database, Ex-pert System	Question Database, Ex-pert System
ActiveMath	Presentation En-gine	Pedagogical Rules, Course Generator	Student Model	MBase, Course Genera-tor
Equation Guru	Interface Module	Pedagogical Knowledge, Stu- dent Diagnostic Module	Student Model, Student Diagnostic Module	Domain Knowledge Base, Do- main Expert Module
A General Architecture for Intelligent Tutoring of Diagnostic Classifi-cation Problem Sol-ving	User Interface	Pedagogical Model	Student Model	Expert Model, Case Da-tabase
An intelligent tutoring system for deaf lear-ners of written English	User Interface	Response Generation Modul, Error Identificati-on Module	User Model	Expert Model, Domain Know- ledge Base
Toward an Intelligent Tutoring System for teaching Law Stu-dents to argue with cases	User Interface	Pedagogical Module	Student Model	Domain Expert
KerMIT/SQL Tutor	Interface	Pedagogical Module, Cons- trained Based Modeller	Student Modells	Constraints, Problems/ Soluti- ons
VC Prolog Tutor	Lehr-/ Lernumge-bung	Auswahl/ Generierung, Diagnose/ Diagnosewis-sen, Didaktisches Modell	Benutzermodell	Domänenmodell, Übungsspezifikationen, Anwendungsdomänen-modell

Tabelle R-II2g-1: Verwendung der Grundkomponenten in vorhandenen ITSs [2006 OERTEL]

Reference	Expert Knowledge	Pedagogical Know-ledge	Learner Element	User Interface
(Corbett et, all, 1997)	Domain Knowledge	Peagogical Module	Student Module	Poblem Solving Envi-ronment
(Lelouche, 1999)	Domain Expert Module	Tutorial Module (Pe-dagogy Expert)	User Model	User Interface
(Alpert et.al., 1999)	Expert Solver	Tutorial Module	Student Module	User Interface
(Melis et.al., 2004)	MBase (Course Genera-tor)	Pedagogical Rules (Course Generator)	Student Model	Presentation Engine

Tabelle R-II2g-2: Verwendung der Grundkomponenten in vorhandenen ITSs [2006 HARRER and MARTENS]

Domänenwissen (in Englisch: domain knowledge)

Die Komponente „Domänenwissen“ ist eine Datenbank von Wissens welches gelehrt werden soll. Es gehört der Domäne des Lehrstoffs an. Es besteht aus den Konzepten, Regeln, Fakten, Texten, Bildern und Lösungsstrategien welche dem Lerner vermittelt werden sollen. Fachwissen wird als korrekt angesehen und kann als Vergleich zum Auswerten der Antworten des Lernenden herangezogen werden. So kann Wissen abgerufen, verändert oder hinzugefügt werden. Die Komponente kapselt die Funktionalität wie das zu lernende Wissen in Daten auf dem Dateisystem umgewandelt und gespeichert wird. Das Wissen ist damit strukturiert und leicht verständlich zugänglich in einem gegebenen für das System bekannten Datenformat.

In der umgekehrten Richtung werden die Daten aus dem Speicher geladen und in ein verständlicheres Format zurückkonvertiert und wieder ausgegeben.

Lernermodell (in Englisch: student model)

Student Model (im deutschen als Lernermodell bezeichnet) ist die Komponente, die das Wissen über den Fortschritt des Lernenden verwaltet. Das Wissen aus dem Lernermodell benötigt zur Auswertung häufig das Domänenwissen. Im Lernermodell wird aufgezeichnet, was der Lernende gemacht hat. Dies wird ausgewertet um einen Lernzustand des Lernenden zu modellieren. Es kann den aktuellen Wissensstand vom Domänenwissen modellieren. Auch gehört es mitunter zu dessen Funktionalität die Entwicklung, den Fortschritt und Präferenzen des Lernenden zu modellieren. Manchmal ist das Lernermodell der Verlauf der besuchten Lehrinhalte, der korrekt oder falsch beantworteten Fragen oder nur eine Zahl welche den Gesamt- oder Durchschnittswert speichert. Die Informationen über den Lernenden lassen sich beliebig genau speichern, bis hin zu einer Gewichtung jeden Inputs abhängig von Zeit, Quantität und Qualität. Genau wie die Domänenwissenkomponente muss die Konvertierung zur Speicherung des Wissens komplett von dieser Komponente für das restliche System gekapselt werden.

Pädagogikmodul (in Englisch: pedagogical knowledge)

Pedagogical Knowledge - im deutschen als Pädagogikmodul bezeichnet - ist die dritte Komponente welche Wissen verwaltet und den Zugriff und dessen Speicherung abkapselt. Es gibt zwei Arten wie diese Komponente bisher eingesetzt wird. Einmal als ein Komponente die Verhaltensvorschläge enthält, zum Beispiel wenn ein Lernender gewisse typische Fehler wiederholt, wie man am besten vorgeht. Das dann in dieser Variante der Komponente gespeicherte Wissen beinhaltet das didaktische Wissen für das Domänenwissen um effizient zu unterrichten. Für die Nutzung des Wissens aus dieser Komponente ist das Wissen aus dem Lernermodell für das ITS sinnvoll, weil dann nicht nur die allgemeinen Verhaltensregeln für die Didaktik gelten, sondern auch angepasst vorgeschlagen werden kann wie sich bei einem bestimmten Typ von Lernenden verhalten werden soll. Dazu muss erst das Wissen aus dem Lernermodell ausgewertet werden, um die passende Lehrstrategie für den Lernenden zu verwenden. Es gibt aber auch eine zweite, mindestens genauso häufig implementierte Bedeutung, die dieser Komponente viel mehr Funktionalität zuteilt. Diese enthält dann häufiger das Wort „Modul“ („module“) im Namen. Es übernimmt dann andere oder zusätzliche komplexere Aufgaben, unter anderem die Auswertung von dem Lernermodell, die Auswertung von der Kombination von Pädagogikwissen und Wissen aus dem Lernermodell. Es beinhaltet dann meist die Aufgabe Entscheidungen darüber zu treffen welcher Lehrstoff als nächstes passend für den Lernenden ist. Diese Funktionalität kann simpel gehalten werden, wie zum Beispiel dass nur vom Lernenden selten oder gar nicht gesehenes Lehrmaterial priorisiert wird. Komplex programmiert hat diese Komponente viele Funktionen und übernimmt die Adaption des Lehrmaterials an den Lernenden. Das Pädagogikmodul mit mehr Funktionalität als reinen Wissenszugriff kontrolliert den didaktischen Ablauf und bestimmt damit auch den Prozess. Es kann auch ein Lehrziel mit Lehrstrukturen welche sich als didaktisch sinnvoll erwiesen haben besitzen und um dieses Ziel zu erreichen braucht es sowohl Zugriff auf das Domänenwissen als auch auf das Lernermodell um damit eine Reihenfolge von Lehrinhalten passend für den Lernenden zu erstellen (Sequenzialisierung der Lehrmaterials). Die Anwendung dieser Komponente ist also sehr unterschiedlich und lässt sich vielleicht darauf zurückführen, dass diese Komponente in der Historie der ITS-Softwarearchitekturen relativ spät integriert wurde und es heute zwar grundlegend für ein ITS ist, aber unbekannt für vorherige Systeme (siehe Kapitel „Geschichte der Lern-/Lehrsysteme hin zu ITS“).

Benutzerschnittstelle (in Englisch: user interface)

Die Benutzerschnittstelle ist die Schnittstelle mit dem der Nutzer mit dem ITS interagiert. Der Nutzer ist hauptsächlich der Lernende, allerdings können andere Nutzerrollen auch vorhanden sein. Darunter fallen der Autor und die Aufsichtsperson (Betreuer/Supervisor). Dies ist die Komponente welche die Interaktionen der Lernenden aufnimmt und Informationen zurück übermittelt. Die Eingabe des Lernenden ist in der Regel die Tastatur und die Maus. Die Ausgabe von Informationen an den Lerner erfolgt auf dem Bildschirm und den Lautsprechern. Die Interaktion des Lernenden mit dem ITS wird häufig durch eine Benutzeroberfläche gestaltet, eine Textkonsole war nur in älteren Systemen vorhanden. Eine gute Benutzerführung und Benutzerfreundlichkeit fördert die Interaktionen des Lernenden, wie auch seine Motivation mit dem ITS zu lernen.

Zusammenfassung

Ein ITS ist mehr in dem Kognitivismus angesiedelt, es gibt allerdings einen fließenden Übergang zwischen den einzelnen Theorien, sodass eine Reichweite von ITSs existieren welche mehr in Richtung Behaviorismus oder Konstruktivismus gehen und trotzdem hauptsächlich noch dem Kognitivismus zugeteilt werden können. Es gibt auch ITS die auf dem Konstruktivismus aufbauen. Doch desto weiter diese sich entfernen um so schwerer fällt es diese noch als ITS zu bezeichnen. Die Architekturen der Anfangszeit konzentrieren sich darauf, welche wenigen Komponenten existieren und nicht deren klare Verbindungen zueinander. Deshalb wurde hier auch nur auf die Funktionsweise der Komponenten eingegangen. Diese vier Komponenten sind die Komponenten welche in wissenschaftlichen Arbeiten bisher am häufigsten herausgearbeitet wurden als die Notwendigen Softwareteile ein ITS zu entwickeln. Mit diesem groben Aufbau von vier notwendigen Komponenten war es möglich sehr stark variierende ITSs zu einwickeln und deren Auslegung durch den Funktionsumfang der einzelnen Komponenten zu bestimmen. Dies ist besonders bei dem Pädagogikmodul sichtbar, dessen Einsatzzweck stark unterschiedlich ausgelegt wird.

II.2.h. Vorhandene ITS-Softwareframeworks

Softwarearchitekturen sind sehr hilfreich als eine Hilfestellung für eine Implementierung eines neuen ITS. Sie geben dem Entwickler aber keinen verwendbaren Code. Softwareframeworks hingegen sind Code, allerdings bieten sie nur einen geringeren Teil der benötigten Arbeit als brauchbaren Code an. So hat man mit Softwareframeworks zwar bereits einen guten Anfang an dem man weiterentwickeln kann, bindet sich jedoch an eine festgelegte Technologie und deren Entscheidungen. Wenn man jetzt von einem abstrakten Architekturmodell zu einem fertigen ITS möchte, so müsste man ohne Softwareframeworks alles von Grund auf neu programmieren. Eine teilweise in der jeweiligen Programmiersprache definierten Spezifikation, einzelne halbfertige abstraktere Klassen, komplette wiederverwendbare Bausteine oder gar ganze Softwareteile, bei denen man sich jeweils der technologischen Vor- und Nachteile bewusst ist, sind da sehr hilfreich.

Es gibt eine allgemeine abstrakte Beschreibung eines Frameworks für ITS in [2003 LELOUCHE and LY]. Diese versucht ein Framework ohne Software zu beschreiben, also eine Anleitung wie man ein ITS nach Architekturvorgabe entwickelt. Es hat die folgenden funktionellen Anforderungen beschrieben.

- Es muss eine allgemeine high-level architectural Ansicht liefern
- Es muss eine allgemeinen Kommunikationsmechanismus zwischen den einzelnen Komponenten spezifizieren
- Es muss eine allgemeine Wissensrepräsentation für jeden Wissenstypen liefern
- Es muss eine große Menge an Methoden für jede Komponente liefern
- Es muss die Flexibilität gewährleisten, vorgefertigte Komponentenfunktionalitäten zu erweitern und zu ändern

Ein anderes Framework welches sich auf die Erkennung und Analyse von Emotionszuständen der Lernenden spezialisiert hat ist EMASPEL (es steht für Emotional Multi-Agents System for Peer-to-peer E- Learning) [2008 NEJI, AMMAR and ALIMI], siehe Abbildung R-II2h-1. Dabei gibt es kein formales Architekturmodell, sondern es wird beschrieben wie man mittels von Agenten nebenläufig stetig ein aktuelles emotionales Profil über den Lernenden hat. Es ist ein Softwarewerkzeug mit kaum abstrakten formalen Definitionen, dessen Ideen über den Einsatz von Agenten nicht genügend abstrahiert wurden. Wie man in Abbildung R-II2h-1 erkennen kann, ist die Übersicht dieses Frameworks eine sehr konkrete Benutzeroberfläche der Software.

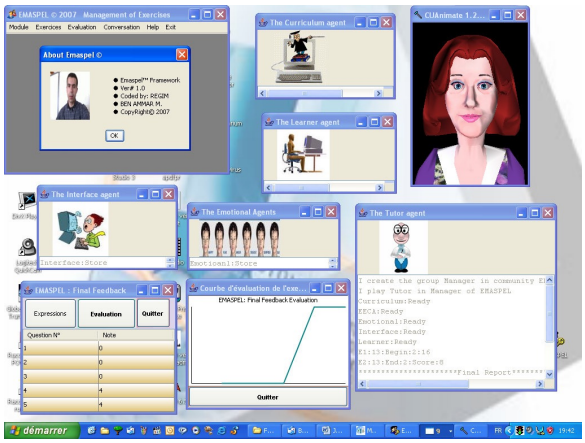


Abbildung R-II2h-1: EMAPSEL Framework [2008 NEJI, AMMAR and ALIMI]

Für eine direkte Entwicklung ist ein programmatisches Softwareframework besser geeignet, weil es für den Entwickler genau soviel an Vorentwicklung bietet wie möglich, aber dabei die Freiheit lässt die Oberfläche, Softwareabhängigkeiten und Datenbanken selbst zu bestimmen. Es gibt wie im ersten Abschnitt dieser Arbeit eine klare Trennung zwischen dem abstrakten Architekturmodell und einer Implementierung. Die Implementierung eines Softwareframeworks hält sich zwar an eine Spezifikation wie dem Architekturmodell, erfordert aber im Gegensatz zu einem abstrakten Framework auch Entscheidungen zu treffen um die Abstraktion in eine konkrete Implementierung umzuwandeln. Das Softwareframework stellt keine ganze Ebene zwischen Modell und Implementierung dar.

Der zentrale Punkt der Konkretisierung ohne Vollendung einer Softwarearchitektur geschieht durch eine Entscheidung wie und mit welchen Mitteln man implementiert. Man konkretisiert dabei nicht alles, viele Entscheidungen müssen für ein Softwareframework offen bleiben. Es gibt noch genügend unterschiedliche Entscheidungen die getroffen werden müssen, welche absichtlich nicht in der Architektur spezifiziert sind. Die Entscheidungen sind folgend aufgezählt.

- Welche Programmiersprache ein Softwareframework verwendet
- Welche Plattformen und Betriebssysteme unterstützt werden
- Mit welchen Mustern die einzelnen Bereiche des Architekturmodells umgesetzt werden

Das Softwareframework stellt keine ganze Ebene zwischen Modell und Implementierung dar. Es ist keine komplette Konkretisierung des Architekturmodells, sondern ein Werkzeug um konkrete Vorgaben in einer Programmiersprache umzusetzen. Es kann nicht immer alles von der Spezifikation in der jeweiligen Programmiersprache umgesetzt werden. Deshalb ist es auch immer nötig, dass sich Entwickler nicht nur auf das Softwareframework verlassen.

Bei der Programmiersprache gibt es viele wichtige Faktoren die die Entscheidung beeinflussen. Es ist ein Für und Wider um abzuwägen, welche Programmiersprache passend ist. Eine Programmiersprache kann auf mehr Plattformen verfügbar sein und bildet die Spezifikation teilweise schlechter ab weil bestimmte generische Vorraussetzungen nicht

beschrieben werden können. Eine andere möglich wichtige Eigenschaft ist die Beliebtheit und Verbreitung einer Programmiersprache, weil eine unbekanntere und/oder unbeliebtere Programmiersprache in der ein Softwareframework programmiert wurde dazu führt das es weniger verwendet wird. Programmiersprachen in der weniger Programmierer vorhanden sind und schwer zu erlernen sind oder welche die sehr speziell sind für ein Gebiet, zum Beispiel Haskell. Letztendlich gibt es kein perfektes Softwareframework für ein Modell. Jede Entscheidung führt zu einer Konkretisierung welche das Softwareframework genau wie auch bei einem ITS in einigen Fällen besser macht und in anderen schlechter. Sinnvoll ist die Fälle zu bevorzugen welche eher eintreten, „JaBInt“ (vorheriger Name war „JaBIt“).

JaBInT ist ein frei verfügbares in Java geschriebenes ITS-Softwareframework. Es wurde in einer Masterarbeit an der Universität Rostock entwickelt [2006 OERTEL]. Es wurden für viele kleinere Lehrsysteme an der Universität eingesetzt. Das Softwareframework „JaBInt“ wird nicht mehr weiterentwickelt, legte aber als Machbarkeitsstudie einen guten Grundstein. Die Architektur selbst wurde in einer Game-based Architektur weiterverwendet und wird in diese Richtung auch weiterentwickelt. Die Flexibilität des ITS nach der Implementierung steht im Vordergrund. Die Wiederverwendbarkeit von Code, die durch die Gleichartigkeit der Grundarchitektur ermöglicht wird, ist auf die Sprache Java beschränkt. Das Modell ist sehr allgemein gehalten und schreibt wenig im Verhalten vor. Eine schwache formale Spezifikation könnte ohne das Softwareframework oder Javakenntnissen zu Fehlinterpretationen führen. In der Architektur, zu sehen in Abbildung R-II2h-2, sieht man die häufigsten Grundkomponenten von ITS, bei dem jede Komponente aus Modulen besteht. Module oder ganze Komponenten sind auswechselbar.

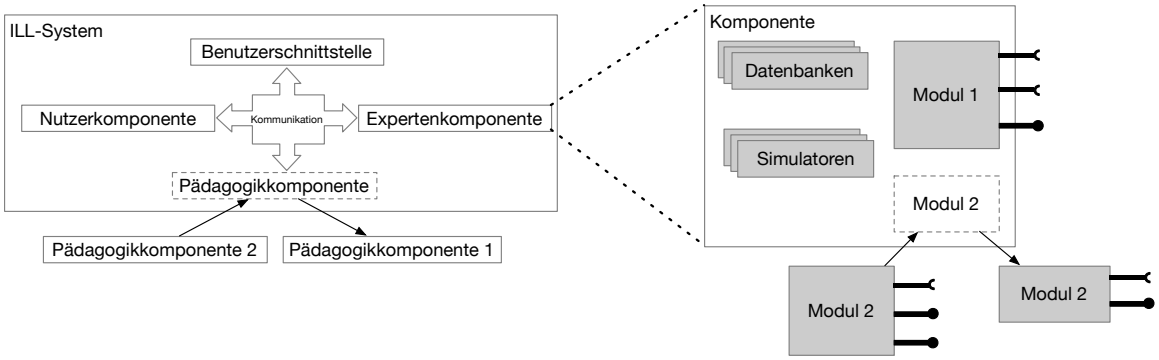


Abbildung R-II2h-2: Architektur von JaBInt [2006 OERTEL]

Cognitive Tutor Authoring Tools CTAT

CTAT ist ein frei verfügbares, von Domänen unabhängiges ITS-Softwareframework, welches in der Carngie Mellon Universität entwickelt und weiterentwickelt wird. Deren Ziel ist es die Kosten für die Entwicklung von ITSs zu verringern [2010 PAQUETTE, LEBEAU and MAYERS]. Es gibt eine jährlich aktualisierte Version auf der Webseite [2018 CARNEGIE MELLON UNIVERSITY]. Dieses Softwareframework spezialisiert sich darauf, das Verhalten eines ITS anhand zweier Modelle zu implementieren, entweder als ein kognitiver oder als ein „Example tracing tutor“. Für einen „Example tracing tutor“ wird für den Lernenden lediglich ein Browser ("Firefox“ oder „Chrome“) und ein Cloudspeicher-Dienst benötigt ("Google Drive“ oder „Dropbox“).

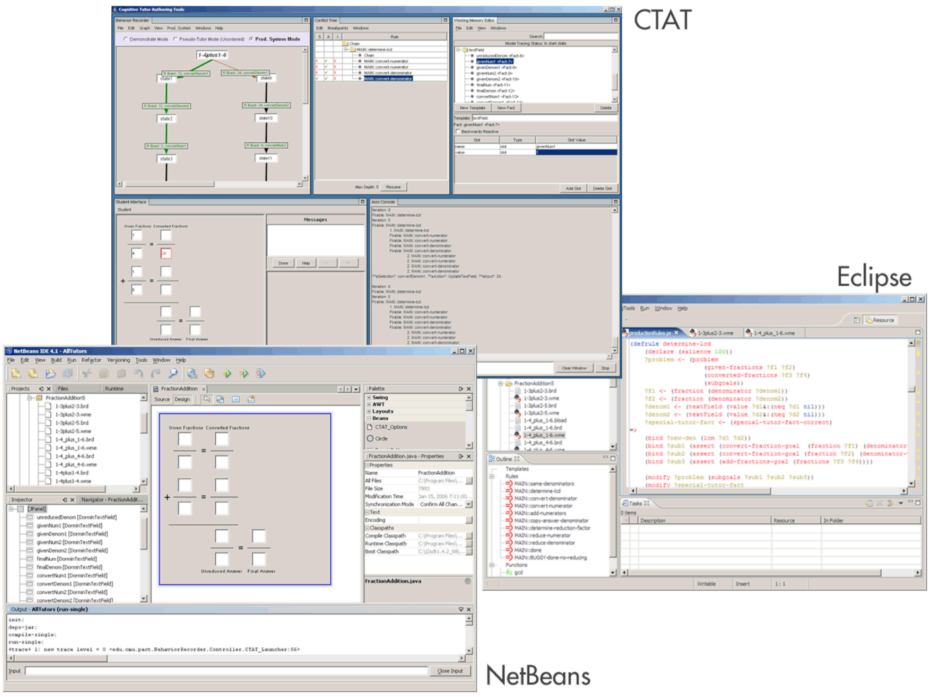


Abbildung R-II2h-3: Softwarewerkzeuge in CTAT [2006 ALEVEN, MCLAREN and SEWALL]

In Abbildung R-II2h-3 sieht man die verschiedenen Softwarewerkzeuge welche benötigt werden um eine Lehrsitzung zu kreieren. Die eigens entwickelten Software ist dabei sehr kompliziert und erfordert zusätzlich „Eclipse“ und „NetBeans“. Beides sind komplexe Entwicklungsumgebungen für Programmierer und fordern dementsprechend Hintergrundwissen in Informatik. Die Verwendung von CTAT hat viele Nachteile, es bindet an die von CTAT gewählten Technologien/Werkzeuge (wie zum Beispiel Java, Flash, Eclipse, NetBeans und so weiter). Eine Weiterentwicklung von drei Werkzeugen die zusammen genutzt werden ist aufwändig. CTAT entbindet nicht von Informatikhintergrundwissen und CTAT selbst ist nicht intuitiv genug um von Laien verwendet zu werden. Der Vorteil hingegen ist eine beschränkte Sicht auf ITSs, die es wahrscheinlich erlaubt eine gewisse Art von ITS wesentlich schneller zu entwickeln. Angewendet sieht man „CTAT“ in einer Architektur von „Mathtutor“ in Abbildung R-II2h-4. Mathtutor wurde von den gleichen Personen entwickelt wie CTAT selbst. Das untere graue Rechteck in der Abbildung mit der Bezeichnung „CTAT Authoring Tools“ ist die eingebundene CTAT-Architektur, die oberen grauen Rechtecke in der Abbildung bestehen aus einer Client-Server-Architektur des implementierten ITS. Die zwei Bereiche bei denen CTAT in diesem Fall hilft sind, dass Verhalten des ITS zu bestimmen und ein Werkzeug für das Erzeugen von Benutzeroberflächen bereit zu stellen. Die Serverseite von Mathtutor hat trotz CTAT viele Datenbanken, die Lernendenverwaltung und den „Tutoring Service“ zu stellen.

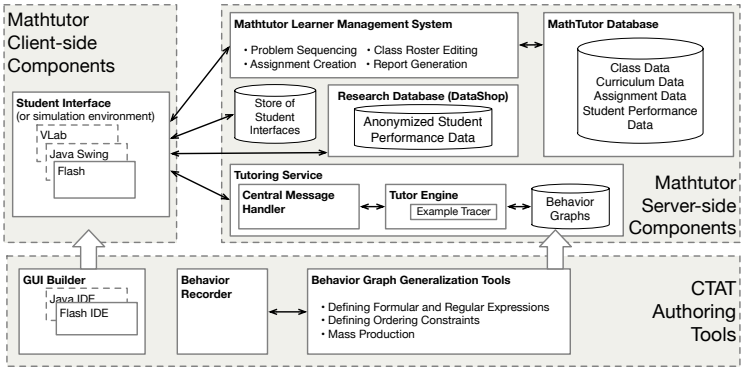


Abbildung R-II2h-4: CTAT in einer ITS-Architektur [2009 ALEVEN, MCLAREN and SEWALL]

ASTUS

ASTUS ist das am weitesten fortgeschrittene allgemeine ITS-Softwareframework. Ziel der Entwicklung war, die Implementationszeit von ITSs zu verringern, indem häufig benötigte Softwarekonstrukte vorprogrammiert sind [2010 PAQUETTE, LEBEAU and MAYERS]. Dabei beschränkt sich das ASTUS-Softwareframework die „Model-Tracing-Tutors“. Es beschränkt sich auch auf Problemlösungsdomänen, welche besonders geeignet für ITSs sind (zum Beispiel Mathematik durch viele formale Regeln). Das Softwareframework benutzt Vorlagen für die Interaktion mit dem Lernenden für Problemlösungsdomänen. Die Wissensmodell ist spezifisch auf genau dieses Softwareframework aufgebaut und beschreibt immer Problem und seine Zwischenschritte um dieses zu lösen.

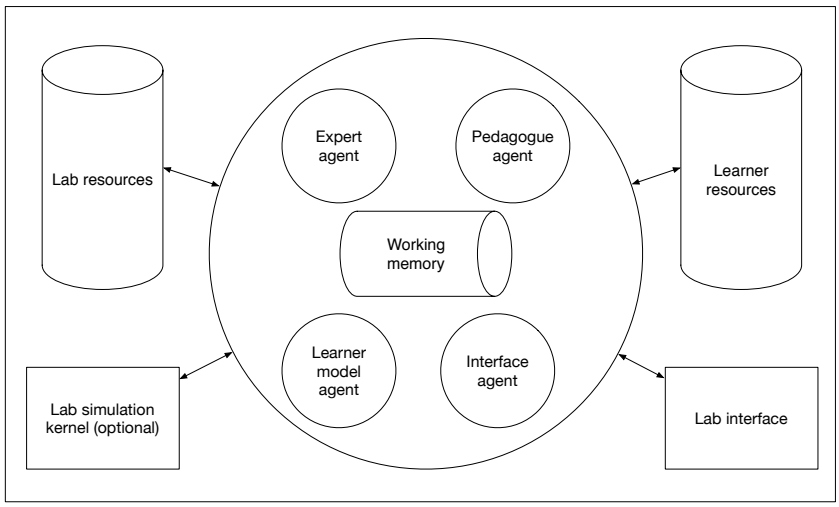


Abbildung R-II2h-5: The ASTUS Framework [2008 FORTIN, LEBEAU and ABDESSEMED]

Im Vergleich ist der Ansatz abstrakter an die Modellierung heranzugehen der Richtige, doch er geht nicht weit genug. Das Modell hinter dem Softwareframework ist nicht formalisiert und vollständig genug für alle Typen von ITS, siehe dazu die Architektur in Abbildung R-II2h-5. Es konzentriert sich sehr stark auf die „Model-Tracing-Tutors“ und die Problemlösungsdomänen die formal genau spezifiziert sind. Zudem fehlt neben der Architektur eine Prozessbeschreibung. Es bleibt bei einem guten Softwareframework, welches als positive Ausnahme einen Debugger besitzt um Fehler zur Laufzeit zu finden. Trotz dessen sollte das Softwareframework, laut Aussage der Entwickler, nur in komplexen und genau definierten Aufgaben stattfinden.

II.3. Vorüberlegungen

II.3.a. Einordnung von ITSs in Lehr-/Lernsystemen

Ein Lernender kann seine Lerneffizienz steigern indem er Hilfsmittel verwendet. Die klassischen Werkzeuge hierfür sind Bücher, Stift und Papier. Der Computer hat auch hier Einzug gehalten und kann als weitere Werkzeuge in Form von Software bereit stellen. Die verschiedenen Softwarearten die von Lernenden verwendet werden können sind in Abbildung R-II3a-1 aufgeführt. Es gibt die Lehrsoftware, die Verwaltungssoftware und die Werkzeugsoftware. Verwaltungssoftware ist für die Organisation, Verwendung und Bereitstellung von Lehrmaterial da. Werkzeugsoftware sind bloß Programme die sich für bestimmte Aufgaben beim Lehren eigenen, an sich aber nicht die Aufgabe haben zu lehren. Diese beiden Arten sind aus gerade diesem Grund nicht im Fokus dieser Arbeit. Lehrsoftware hingegen versucht auf unterschiedlichen Wegen Wissen zu vermitteln. Jedes Programm, welches sich zu einer Software zum Lernen zählt, gehört zu E-Learning-Software, selbst wenn es keine Lehrsoftware ist. Der Begriff ist schwammig geworden und wird daher mit jeglicher Software in Verbindung gesetzt, dass explizit für das computergestüzte Lernen gemacht wurde. Werkzeugsoftware wird häufig nicht zur E-Learning-Software gezählt, da ihr Einsatzzweck nicht nur auf die Lehre ausgelegt ist. Verwaltungssoftware für Lehrsoftware ist besonders häufig in Universitäten vertreten und wird gerne zum E-Learning gezählt, obwohl deren Funktionsumfang eher unterstützend für Dozenten ist. Die ganze Software zum Lernen steht unter dem Schlagwort „Technology Enhanced Learning“ (TEL) als der Oberbegriff für das Lernen mit Technologien, der aber mehr beinhaltet als Software zum Lernen [2015 MARTENS], zum Beispiel „E-Instruction“ für unter anderem „Distance Learning“ als ein Mittel eines Lehrers mit Technologie über eine größere Distanz zu lehren.

Ein ITS kann auch grob beschrieben werden als ein Computerprogramm, welches Domänenwissen besitzt und es (mittels pädagogischem Wissens) lehren kann. Das ITS reagiert auf Aktionen des Benutzers und adaptiert sich an den Benutzer. Ein ITS ist zwar ein Softwaresystem zum erleichtern des Lernens (TEL), allerdings muss man es auch klar abgrenzen von ähnlichen Softwaresystemen, wie zum Beispiel einem Expertensystem, welches nur über Domänenwissen urteilt. Es gibt viele Systeme die mit einem ITS zusammen arbeiten können die selbst keine Lehrsoftware sind. Bei diesen kann man überlegen ob sie in ein ITS integriert werden können um die Anzahl der benötigten Werkzeuge reduziert.

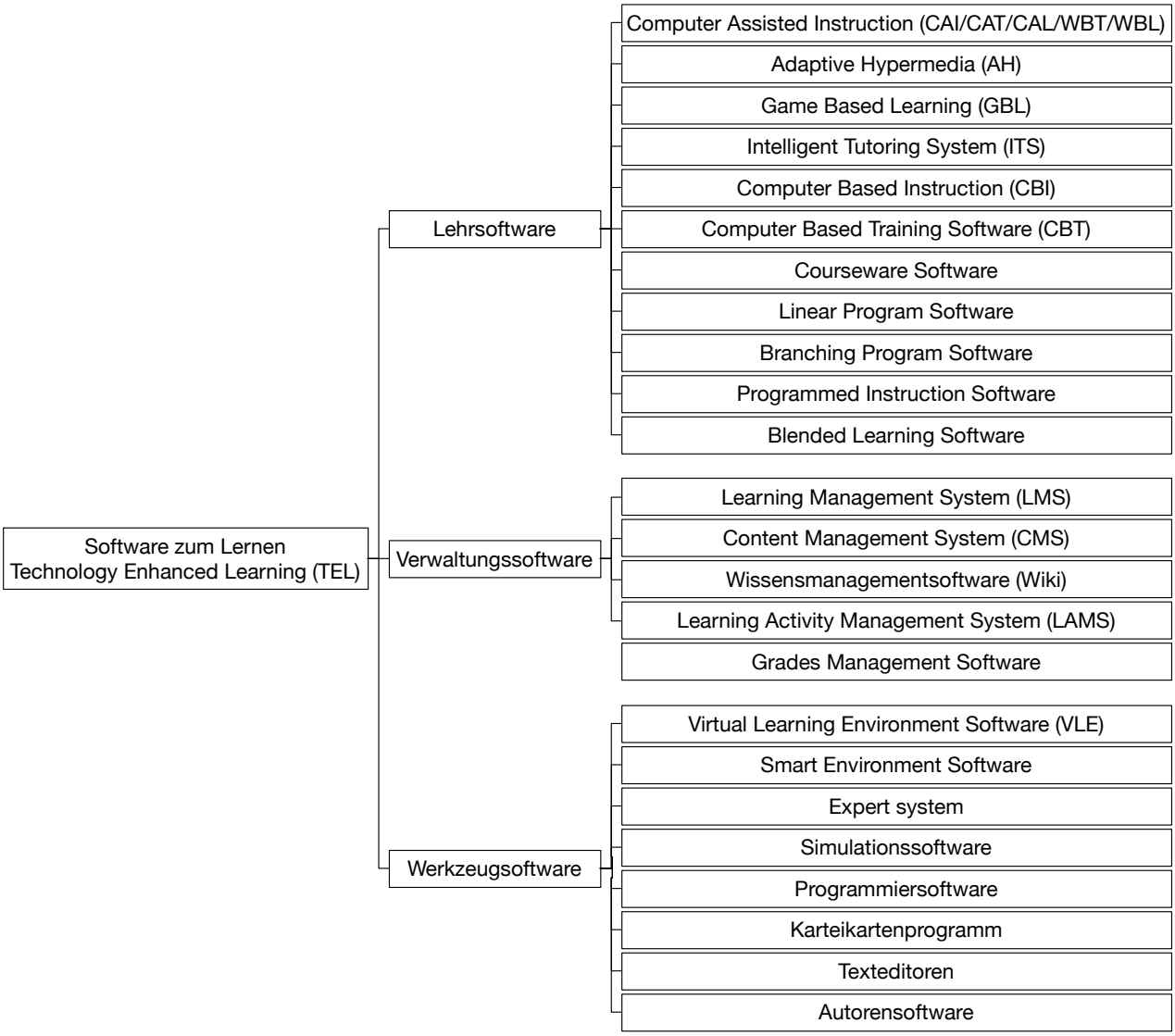


Abbildung R-II3a-1: Taxonomie von Software zum Lernen inklusive Abkürzungen

Die Entwicklung der verschiedenen Systemtypen laufen unabhängig voneinander ab. Die Stärken und Schwächen sind für die meistgenutzten Anwendungsfälle der jeweiligen Nutzer angepasst. Die Systeme könnten theoretisch Domänenwissen, dazu passendes pädagogisches Wissen und bereits bekanntes Wissen über Lernenden untereinander austauschen. Praktisch benutzen sie unterschiedliche definierte Strukturen die inkompatibel mit den anderen sind. Die Kategorie „Lehrsoftware“ hat sehr breit gefächerte Einsatzzwecke und sind teils für mehrere Expertendomänen einsetzbar. Andere Arten von Lehrsoftware verbinden Lernende und den Lehrinhalt weniger stark als ein ITS und erfordern auch eine geringere Analyse von beiden. Die ITSs gehören zu einer Untergruppe der Lehrsoftware welche eine Adaptivität zum Nutzer besitzen. Darunter gehört auch Software der Klasse „Adaptive Hypermedia“ (AH). Ein AH besitzt ein „expert knowledge model“ und ein „student model“ mit ähnlichen Einsatzzweck wie bei ITSs [1998b BRUSILOVSKY]. Dennoch ist die ausgereifte Analyse und KI der ITS weitaus höher angesetzt als die der AH, welche mit ihren statischen Lehrelementen eher mit der Klasse der Branching Programs zu vergleichen sind. Die Entwicklung der KI war immer ein starker Fokus von ITS [1996 URBAN-LURAIN].

II.3.b. Beteiligte von ITS-Entwicklungen

Es gibt viele verschiedene Personengruppen welche an einem ITS während und nach der Entwicklung beteiligt sind und deren Fähigkeiten in unterschiedlichen Bereichen liegen [2010 PAVLIK and TOTH]. Es gibt viele Personen und unterschiedliche Personengruppen die einen Anteil an der Entstehung und Nutzung von ITSs haben mit jeweils verschiedensten Aufgaben[2010 MARTENS and HARRER]. Die Anforderungen der unterschiedlichen Gruppen ist sehr unterschiedlich. Für die konkrete Entwicklung einer vollständigen ITS-Idee werden Systementwickler benötigt, doch da diese nicht das nötige Hintergrundwissen über die Lehre und den Lehrstoff an sich besitzen, werden Experten benötigt welche das System auf einer abstrakteren Ebene mit entwerfen. So wird auch unter anderem Wissen über den Entwurf von Benutzeroberflächen Psychologen und Designer und Grafiker benötigt. Dabei fällt die Umsetzung in die Hände der Entwickler, welche interpretieren und entscheiden wie etwas umgesetzt wird (und manchmal auch, was umgesetzt wird). Die Interpretation kann umso genauer werden, desto mehr etwas definiert und spezifiziert wurde. Beispiele für Personengruppen die zur Entstehung eines ITS maßgeblich beitragen werden folgend aufgezählt.

- Grafiker
- Didaktiker
- Programmierer
- Systementwickler
- Designer für Benutzeroberflächen
- Pädagogen für die gewählte Zielgruppe (unter Umständen Lehrer oder Lehrforscher)
- Psychologen für die Motivation durch das System
- Experten der Wissensdomäne die gelehrt werden soll (unter Umständen Lehrer)

Bei den Nutzern von ITSs gibt es zwei große Gruppen, die Autoren und die Lernenden. Die Autoren sind zwar Nutzer des Systems, ihnen kommt aber eine spezielle Rolle zu. Sie sind Experten, die Datenbanken des Systems nachträglich anreichern können. Sie sind aber nicht zwangsläufig in dessen Entwicklung miteingebunden. Meist bisher vernachlässigt in einer ITS-Entwicklung sind Autorenfunktionen ein wichtiger Bestandteil um ein ITS mittel- oder langfristig zu verwenden, weil dadurch ein benutzerfreundlicher Weg der leichten Anpassung von einem ITS entsteht. Die Autoren sind Personen die eine Submenge von den Beteiligten der Entwicklung sind. Beispiele sind folgend aufgezählt.

- Lehrer für die Domäne
- Didaktiker mit Wissen in der Domäne
- Psychologen für die Motivation für den Lehrstoffs
- Experten der Wissensdomäne die gelehrt werden soll

Zwar sind Lernende die Hauptzielgruppe auf die das System ausgerichtet werden sollte, dennoch darf die Nutzerfreundlichkeit für Autoren, Betreuer und Lehrer nicht zu sehr vernachlässigt werden. Dabei fällt die Aufgabe des Lehrens nicht auf die Betreuer, sondern auf das ITS selbst. Personen als Beispiele die dafür verwendet werden sind folgend aufgezählt.

- Tester
- Schüler
- Betreuer
- Studenten
- Probanden für die Forschung

Es sind nicht nur die Einzelpersonen relevant, welche das System als Nutzer verwenden, sondern auch Gruppen von Lernende, sowie Institutionen die solch ein System im größeren Stil einsetzen, zum Beispiel die folgend aufgezählten.

- Kindergärten

- Schulen
- Berufsschulen
- Universitäten und Hochschulen
- Weiterbildungsstätten für Berufe
- Experten aus Verlag, Vertrieb, Produktion

Aber nicht nur die Verwendung sondern auch die Beteiligten dürfen nicht außer Acht gelassen werden, welche ein komplexes System ab einem gewissen Niveau betreiben können und die Auslieferung mit Geräten, Installation und Service anbieten könnten. Ein weiteres Augenmerk welcher in dieser Arbeit relevant ist, sind die indirekten Auswertungen der Ergebnisse für die Forschung, sowohl aus der Sicht der Informatik, als auch für die Lehrforschung. Wann sind Daten von einem ITS für diese Bereiche nutzbar.

- Lehrforschung
- Informatikforschung

II.3.c. ITS-Entwicklungsprozess

Eine Neuentwicklung eines ITS ist ein immenser Aufwand, der von ganz verschiedenen Beteiligten getragen wird [2019a GRAF VON MALOTKY and MARTENS]. Aus der Perspektive der vorliegenden Dissertation in der Informatik ist ein ITS eine Software, welche durch einen Programmierer implementiert werden muss. Aus der Perspektive der wissenschaftlichen Forschung in der Informatik handelt es sich um ein Implementierungsartefakt mittlerer Komplexität, das seit vielen Jahren als Grundlage weiterer fachübergreifender Forschungsvorhaben ist und sich im Rahmen der Digitalisierung von Lehre und Lernen neuer Beliebtheit erfreut.

Damit der Softwareentwickler weiß was er implementiert, bekommt er von den Beteiligten eine gewünschte Spezifikation des Systems, darin können Beschreibungen der Funktionalität, Benutzeroberflächendesigns, Anwendungsfallmodellierungen, Softwarearchitekturen, Systemarchitekturen, Klassendiagramme, Systemanforderungen, Performanzziele und so weiter enthalten sein. Handelt es sich bei diesen Beteiligten nicht um Informatiker, sondern um Mitglieder der oben genannten Zielgruppen mit anderem Bildungshintergrund, dann ist diese Spezifikation wenig informatorisch formal. Die Spezifikation erlaubt es damit dem Softwareentwickler selbst zu entscheiden wie und was implementiert wird, ohne Anforderungen zu missachten. Nach der Programmierung des ITS wird dieses häufig durch später hinzu kommende Änderungswünsche immer wieder angepasst. Sollte es sich um ein Forschungsvorhaben der Informatik handeln, so nimmt der Programmierer eine bestimmte Forschungsperspektive ein, beispielsweise Agentenbasierte Softwarearchitektur, und richtet die Software danach aus. Schwierig wird es dann ohne Referenzarchitektur den wissenschaftlichen Nutzen einer Neuentwicklung kritisch und differenziert darlegen zu können.

Softwareentwickler können nach verschiedenen Lebenszyklen der Softwareentwicklung arbeiten. Es gibt mehrere Phasen für alle Modelle. Passend für eine komplexe Software wie ein ITS sind die iterativen Modelle, weil diese dynamisch auf ändernde Anforderungen reagieren können, siehe dazu Abbildung R-II3c-1. Damit wird der Entwicklungsprozess eines ITS als eine fortlaufende Aufgabe verstanden und das ITS verändert sich in jeder Iteration. Dieser Prozess kann auch auf einen größeren Rahmen wie die Spezifikation eines ITS angewendet werden. Somit ist es nicht mehr der Programmierer welcher die Qualität des ITS vollends testen und bewerten kann, sondern dieser kann nur noch die technische Korrektheit überprüfen. Bei einem derart komplexen Thema wie der Lehre mit künstlicher Intelligenz, ist nur die wissenschaftliche Arbeit (zum Beispiel eines Lehrforschers) in der Lage die Qualität des ITS zu verifizieren.

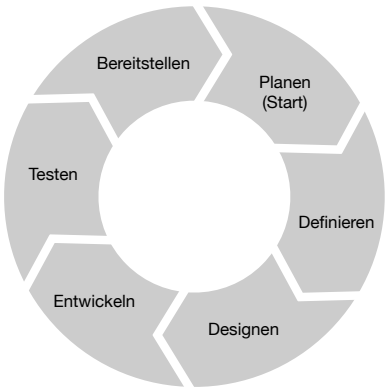


Abbildung R-II3c-1: Software development life cycle elements (SDLC) mit Komponenten aus [2017 SHYLESH]

Für den Bildungsforscher oder den Lernpsychologen sind konstante Umweltbedingungen in Studien essentiell. Da das ITS im Sinne der Bildungsforschung und der Lernpsychologie eine Umweltkomponente darstellt („Lernen mit dem ITS“ als Untersuchungsrahmen) ist es erforderlich, dass zumindest für derartige Forschungsarbeiten das ITS nicht angepasst und geändert wird. Deshalb ist es ungünstig für die Forschungsergebnisse, wenn sich das ITS ständig weiterentwickelt. Vom Forscher muss nur eine Version des gewählten ITS genutzt werden. Weil er nicht die Version ändern kann, muss er warten bis eine für ihn funktional komplette Version zur Verfügung steht und abschätzen ob diese vergleichbar und fehlerfrei genug für seinen Einsatz ist. Die Beobachtung von dem Lehrforscher der Interaktion der Lernenden mit dem ITS kann dann studiert und veröffentlicht werden. Selbst wenn das ITS nochmal in einer Lehrforschungsarbeit verwendet wird, so ist es unwahrscheinlich, dass die Version des wieder untersuchten ITS die gleiche ist. Dadurch, dass die Zyklen der Entwicklung und Forschung so unterschiedlich sind siehe dazu Abbildung R-II3c-2, kann auch nur schwerlich der Forschungszyklus selbst bei nur großen Änderungen in den „Software Development Cycle“ integriert werden. Wünschenswert wäre es, wenn die Weiterentwicklung und Neuentwicklungen der ITSs von den Ergebnissen der Forschungspapieren bestimmt würden.

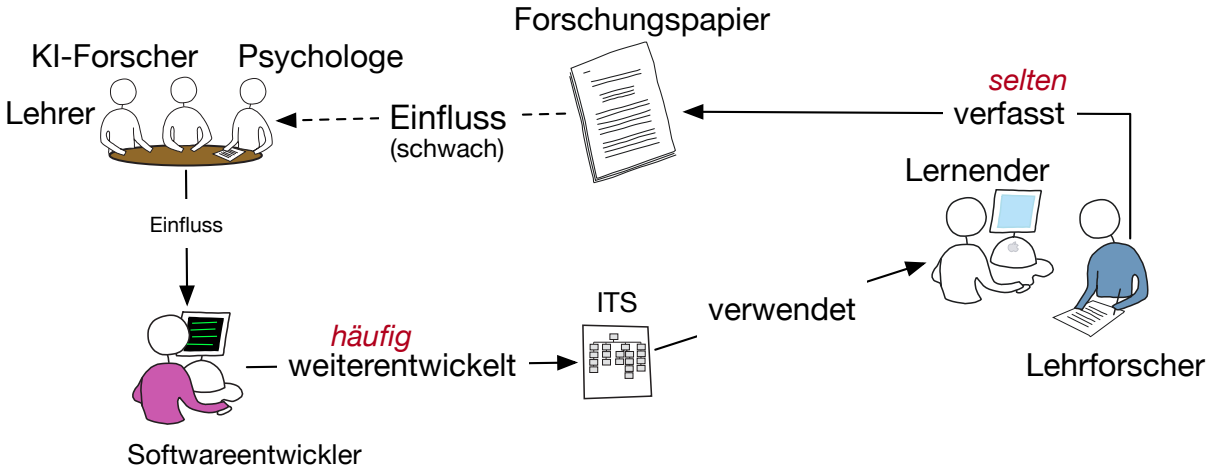


Abbildung R-II3c-2: Einflüsse in der ITS-Entwicklung

Betrachtet man zwei stark unterschiedlich entwickelten ITSs ohne dass diese auf einem Standard beruhen und nutzt sie als Grundlage um jeweils eine Studie durchzuführen, dann führt das zu Problemen. Die Studie wird von Forschenden an Lernenden durchgeführt, welche die Aufgaben und das ITS interpretieren müssen, zusätzlich wird dabei ein Lehrforscher für das jeweilige ITS die Interaktion der Lernenden mit dem ITS auswerten

und interpretieren. Die Ergebnisse der Lehrforschung werden als letzten Schritt in wissenschaftlichen Arbeiten publiziert. Nun sind für die Lehrforschung sehr wichtige Forschungsarbeiten welche Rückschlüsse wegen der Benutzung von unterschiedlichen didaktischen Ansätzen innerhalb der beiden erforschten ITS wenig aussagekräftig, weil die beiden ITSs viele andere Faktoren haben die nicht vergleichbar sind, siehe Abbildung R-II3c-3.

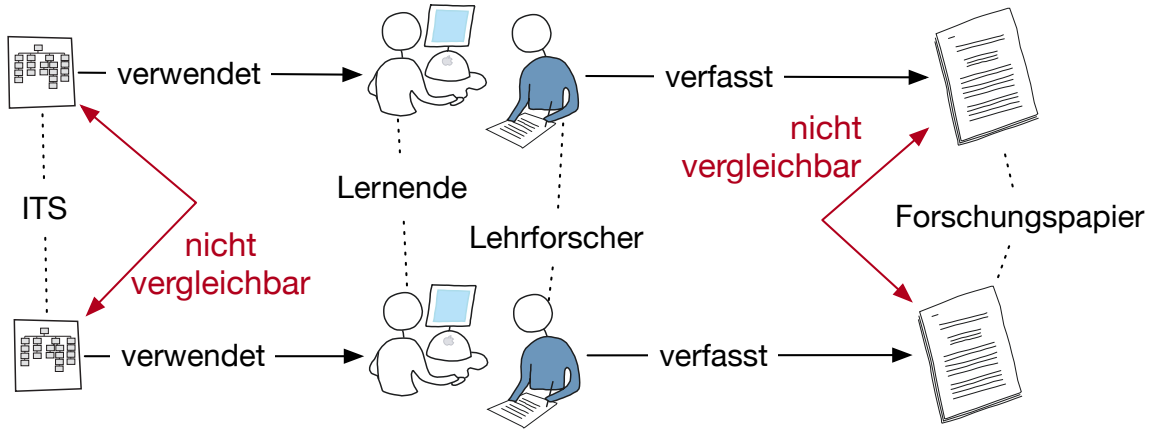


Abbildung R-II3c-3: Vergleichbarkeitsproblem von konkreten ITS

Ohne Einhaltung an einen Standard von den Entwicklern von ITSs beeinflusst dies die Vergleichbarkeit zu vorherigen Forschungsarbeiten negativ. Die meisten ITSs welche entwickelt wurden, halten sich nur sehr grob an Standards. Also entstehen immer mehr neuentwickelte ITSs, welche sich vom Grundaufbau von schon vorhandenen ITSs unterscheiden. Neue Erkenntnisse von Forschungsarbeiten für die Weiterentwicklung oder Neuntwicklung neuer ITSs wird häufig ignoriert oder nicht wahrgenommen und so kommt es zu immer wieder gleichen Fehlern. Ein Standard könnte dieses Problem beheben. Forschungsarbeiten die ihre Fortschritte in einem Standard zusammenführen sind deutlicher wahrnehmbar für Entwickler und es ist weitaus leichter identifizierbar wenn ein ITS einen Standard implementiert oder davon begründet abweicht.

Es sollte auch Ziel von ITSs sein nicht nur verschiedene Softwaresysteme, sondern Unterschiede in der Lehre untersuchen zu können. Dabei ist eine Vergleichbarkeit der zugrundeliegende System wichtig. Interessante Fragen sind dabei folgend aufgelistet.

- Lernen Lernende digital anders als in anderen Lernumgebungen (Psychologisch)?
- Wie erfolgt Lernen überhaupt? Welche Lehrstrategie ist besonders geeignet (für eine Domäne)?
- Wie gut funktioniert Lernen mit digitalen Materialien - funktioniert es besser, schlechter oder gleich gut wie traditionelle Lehr-/Lernformen (Bildungswissenschaftlich)?

Zusätzlich gibt es viele Kommunikationsprobleme von der Entwicklung eines ITS bis zu deren Analyse in Forschungspapieren. Die auch mittels eines Standards der von möglichst vielen Parteien verstanden werden kann reduziert werden könnten. Folgend sind Kommunikationsprobleme erwähnt.

- Kommunikationsproblem zwischen den Interessengruppen
- Kommunikationsproblem von den Interessengruppen zu den Entwicklern
- Kommunikationsproblem von den Lehrsystemen zu den Lernenden
- Beobachtungsproblem von den Testpersonen zu den Lehrforschern
- Verständnisproblem von den Forschungspapieren zu den Lesern

Eines der Ziele der ITSs ist es eine höheren Wissenszuwachs bei den Lernenden zu erwirken als anderen Lehrsysteme, dazu ist es wichtig zu wissen wie die Bildungsforscher arbeiten. Eine typische wissenschaftliche Herangehensweise zur Messung des Wissenszuwachses wird hier aufgezählt.

- Zufällige Auswahl der Probanden nach festen Kriterien
- Herstellen des gleichen Szenarios
- Herstellen einer vergleichbaren Kontrollgruppe
- Modifizierung genau verfolgbarer Entitäten
- Beobachtung und Messung des Lernerfolgs
- Vergleich der Gruppen
- Langfristige Beobachtung

Untersuchungen zum Lehrerfolg eines ITS ignorieren die Softwarearchitektur der Software und damit ebenso die Effekte, die Architekturimmanent sind. Ohne Referenzarchitektur sind bildungswissenschaftliche und lernpsychologische Untersuchungen im größeren Rahmen (zum Beispiel Vergleich zweier Systeme der gleichen Anwendungsdomäne oder der vergleichbaren didaktischen Material- und Supportstruktur) unter Umständen gar nicht möglich. Ein stark unterschiedlicher Aufbau von Funktionalitäten oder Prozessen zwischen den ITSs hat wahrscheinlich Einfluss auf die gemessenen Werte des Lehrforschers. Um die Gesamtheit eines ITS inklusive seiner Funktionalitäten, seinen Aufbau, den verwendeten Lehrablauf zu vereinheitlichen, tragen Standards in der Softwarearchitektur mit Beschreibung der Funktionalitäten und Prozessbeschreibungen bei. Ein weiterer Punkt um die Vergleichbarkeit zu erhöhen ist es, die Benutzeroberfläche und den Aufbau sichtbaren Daten ähnlich zu gestalten. Das bedeutet, dass Benutzeroberflächen ähnliche Datenpunkte anzeigen, ähnliche Menüführung haben, sichtbare Elemente ähnlich aussehen und ähnlich animiert sind. Dies ist aber nicht Fokus der vorliegenden Dissertation und gehört in den Bereich der HCI Forschung („Human-Computer-Interface“).

Noch problematischer wird es wenn der Wissenszuwachs von Lernenden den unterschiedlichen Lehrstrategien zugeschrieben wird, wenn eine Lehrstrategie von einem ITS und eine andere von einem menschlichen Lehrer durchgeführt wurde. Das Verhalten von Lernenden mittels einer Software ist grundlegend anders als mittels eines menschlichen Lehrers. Der Vergleich zwischen zwei ITSs mit gleicher Architektur, Funktionalität, Prozess und so weiter wäre hier konsistenter, wird aber aufgrund des hohen Aufwands gescheut. Es sollte auch kein Vergleich zwischen Implementierungen der ITSs geben, wenn die wissenschaftlichen Arbeiten nur eine Komponente herausnehmen und diese vergleichend betrachten ohne darauf zu achten, ob die implementierten ITSs grundverschieden auch in anderen Bereichen sind.

Um die Probleme in den Griff zu kriegen sind Standards und damit festgelegte Architekturen, wiederverwendbare Komponenten, einem klar definiertem Vokabular, gemeinsame Ontologien die Lösung. Wie folgend in den Punkten von Rodrigues für zukünftige ITSs verlangt [2005 RODRIGUES, NOVAIS and SANTOS].

- Wiederverwendbare Komponenten
- Standarisierungsbemühungen
- Gemeinsames Vokabular
- Ontologien
- ITS-Hüllen
- Verteilte Architekturen
- Personalisierungstechniken
- Fallbasierte Schlussfolgerungen
- Anpassbare mediale Inhalte

Zudem ermöglicht ein vereinheitlichtes Lehrsystem, Annahmen zu untersuchen über die Art wie gelernt wird. Verschiedene ITSs könnten durch höhere Ähnlichkeit in anderen Faktoren (Architektur, Aufbau der Datenbanken oder der Benutzeroberfläche) als den Lehr-

strategien, genau diese besser miteinander vergleichen. Nachträglich kann objektiver analysiert werden, welche effizienter ist. Vergleichbare ITSs sind selten, doch zeigen Forschungsarbeiten, dass grade der Vergleich von Lehrstrategien in ITS eine sehr gute Basis für bildungswissenschaftliche oder lernpsychologische Analysen liefern können. Zum Beispiel wurde untersucht, dass eine restriktiver Ablauf von Aufgaben bei dem erst eine Aufgabe erledigt werden muss bevor eine andere angefangen werden darf, effizienteres Lernen erlaubt, als wenn man es dem Lernenden selbst überlasst jederzeit unfertig zur nächsten Aufgabe zu gehen [2012 MITROVIC and MATHEWS]. Hiroshima, Yamamoto und Wake untersuchten zum Beispiel, dass der Lernende auch dadurch lernt indem er die Aufgaben löst und danach modifizieren muss, sodass er sich der Problemklasse bewusst wird [2010 HIRASHIMA, YAMAMOTO and WAKI]. Solche Forschungsergebnisse wären besser miteinander Vergleichbar und würden eine höhere Relevanz erhalten wenn sie zusammen in einem Bezugssystem wären.

Wissen in Daten zu transformieren ist in ITSs gewiss nicht einheitlich. Es gibt verschiedene Ansätze, welche jeweils auf einem anderen Modell aufbauen. Diese werden benötigt um nicht nur Wissen strukturiert zu speichern und damit effizient zu zu finden sondern auch um den Wissensstand des Lernenden darzustellen. Um das Wissen auch in einem Modell zu repräsentieren gibt es momentan verschiedene Methoden die folgend aufgeführt sind:

- Semantische Netze in CAI-Systemen [1970 CARBONELL]
- Case based reasoning [2005 HAN, LEE and JO]
- Bayesian networks [2002 CONATI, GERTNER and VAN LEHN]
- Productions systems [1990 ANDERSON]

Da es in dieser Arbeit nicht um die Modellierung des Wissens geht, wird darauf nicht tiefer eingegangen. Es sollte nur klar gemacht werden, dass auch hier keine einheitliche Struktur vorhanden ist und die Vielfalt der Modelle in jeder Komponente berücksichtigt werden muss.

II.3.d. ITS-Forschungsbereiche

Es wurden in der Analyse in dieser Doktorarbeit drei Forschungsbereiche für die Forschung an ITSs gefunden [2019a GRAF VON MALOTKY and MARTENS].

Forschungsbereich 1: Grundlagenforschung

Der erste Bereich ist die Grundlagenforschung für ITSs und versucht Fragen zu beantworten, welche die Basis für die anderen Bereiche legt. Zwar ist der erste Forschungsbereich nicht auf detaillierte Implementierung aus, kann aber kleinere Prototypen bereit stellen um einen Nachweis für die Machbarkeit zu liefern. Fragen wie die Folgenden beschreiben den Bereich gut.

- Wie ist die Architektur?
- Was ist historisch wichtig?
- Wie kann man ein ITS beschreiben?
- Wann sollte man ein ITS einsetzen und entwickeln?
- Was sollte man Allgemeinen bei ITS berücksichtigen?
- Welche Modelle eigenen sich zum beschreiben von ITSs?
- Welche Personen sind wichtig und beteiligt an der Entwicklung?
- Welche Prozesse gibt es innerhalb und außerhalb der ITSs und deren Verwendung?

Forschungsbereich 2: Implementierung

Der zweite Bereich implementiert viel mehr als nur kleinere Prototypen. Hier wird sich auf eine Expertendomäne fokussiert um dann ein ITS mit tiefem Verständnis für diese Domäne zu erzeugen. Es erfordert eine gute Zusammenarbeit mit der Lehrforschung in einem Bereich um die am besten passende Lehrstrategie auf die gewählte Domäne anzuwenden. Es wird die Spezifikationen und kleine technische Details festgelegt, und wenig allgemein gehalten. Diese ITSs werden auf ihre Korrektheit validiert und in Studien empirisch ausgewertet. Folgende Fragen geben einen Einblick in den Forschungsbereich 2.

- Was ist praktikabel?
- Was ist technisch sinnvoll?
- Wie gut ist ein spezielles ITS?
- Wird es im Einsatz angenommen?
- Wie bildet man das Wissen einer Domäne ab?
- Welche Technologien sollten verwendet werden?
- Was ist ein guter Ansatz für eine bestimmten Anwendungsfall?

Forschungsbereich 3: Verbessern

Der dritte Bereich analysiert die vorhandenen (bereits implementierten) ITSs. Hauptaufgabe dieser Forschung ist es Schwächen zu erkennen und die besten Techniken für ITSs herauszufinden und zu verfeinern. Es wird geforscht um Einsatzmöglichkeiten zu erweitern, Anpassungen an ITSs oder deren Umgebung vorzunehmen um eine einfache Integration von ITSs in die Lehre zu erlauben. Dazu ist es notwenig, dass möglichst viele gute ITSs im Einsatz sind, damit die Daten über deren Verwendung gesammelt werden können. Diese folgenden Fragen können sich Forscher im Forschungsbereich 3 stellen.

- Wie kann man kosten senken?
- Welche Methode ist effizienter?
- Wie kann man die Zielgruppe besser erreichen?
- Was ist die beste Umsetzung einer Methode?
- Was wird und hat sich langfristig durchgesetzt und warum?

Zielsetzung für diese Arbeit

Ziel dieser Arbeit ist die Schaffung eines Referenzmodells, das im Sinne eines Pre-Standards von der wissenschaftlichen Community im Bereich der ITS Forschung kritisch reflektiert und untersucht werden kann. Mittel- oder langfristig besteht der Anspruch, dass das Modell als Grundlage für die Entwicklung eines Standards genutzt werden kann. Leider gibt es aktuell in der Community keine Standardisierungsbestrebungen, so dass dies nicht mehr im Rahmen der vorliegenden Arbeit erreicht wird. Es entstehen bei einer so abstrakten Modellierung in den Forschungsbereichen verschiedene Grade der Abstrahierung, diese werden folgend zum Verständnis aufgezählt. Ziel ist der Abstraktionsgrad 2.

- Abstraktionsgrad 1 - Softwaremodell abhängig von der Domäne und/oder der Implementierung: Ein Softwaremodell als Modell von einem konkreten System in einer spezifischen Domäne, zum Beispiel ein SQL-ITS als ein Softwareartefakt welches wenig abstrahiert in UML („Unified Modeling Language“) ausgedrückt wurde um als Modell für ein andere ITS zu dienen, welche die gleiche SQL-Domäne und ähnliche Technologien verwenden.
- Abstraktionsgrad 2 - Generelles Modell eines Softwaretypen: In diesem Grad wird die Software unabhängig von möglichen Technologien und Implementierungen betrachtet. Es ist eine Referenz für eine Software über einen Einsatzzweck, zum Beispiel eine Vorgabe wie man ein ITS bauen sollte wenn er auf dem Kognitivismus beruht.

- Abstraktionsgrad 3 - Software in einem System: Ein Abstraktionsgrad bei dem nicht nur Software betrachtet wird. Ein allgemeiner Standard für unterschiedliche Systemarten, wie zum Beispiel eine Systemarchitektur für E-Learning-Systeme, und welche Hardware und Software in welcher Form zusammenarbeiten muss, damit sie funktioniert.
- Abstraktionsgrad 4 - Software als einzelnes Artefakt: Ein Modell bei dem Software nur eine von verschiedenen Teilen ist, das benötigt wird, ohne genauer auf deren Funktionsweise einzugehen. Es kann zur Planung und Verwaltung des größeren Ganzen verwendet werden, zum Beispiel Planung eines Blended-Learning-Unterrichts bei dem Software nur ein Teil von vielen ist, neben der Hardware, dem Lehrer, dem Klassenraum und so weiter.

II.3.e. Vorhandene ITS-Typen

Wenn der volle Funktionsumfang in einem grundlegendem ITS angeboten wird, dann gibt es trotzdem unterschiedliche Wege die Kernfunktionalität zu implementieren. Die Typen von ITSs werden folgend aufgezählt. Dabei ist dies zwar eine wichtige Entscheidung aber so spezifisch, dass sie weder in einer allgemeinen Softwarearchitektur, noch in einem allgemeinen Softwareframework festgelegt werden sollte. Denn keine dieser Methoden, für den Umgang mit eines der schwierigsten Teilen für ITS, ist in einem Großteil der Anwendungsfälle allen anderen überlegen. Es hängt davon ab was genau in einem ITS und in welcher Domäne es umgesetzt werden soll. Die ITS-Vorgehensweisen welche am meisten vertreten sind, sind die Model-Tracing-Tutors und Example Tracing Tutors, zweitrangig einige Constraint Based Tutors.

Model-Tracing-Tutor (MTT)

Man simuliert mit dieser Methode den Lösungsalgorithmus eines menschlichen Experten wenn er diese Aufgabe bekommen würde. Das ITS muss also in der Lage sein die von ihm gestellte Aufgabe selbst Schritt für Schritt lösen zu können und damit einen idealen Lösungsweg zu generieren, den der Lernende bei perfekter Lösung genau so auch macht. Der Lösungsweg besteht für das ITS aus Einzelschritten die als mögliche Aktionen für dem Lernenden bereit gestellt werden. Wenn der Lernende nicht die Einzelschritte macht, welche das ITS gemacht hätte, dann wird davon ausgegangen, dass der Lernende Hilfe benötigt. Das ITS muss bei dieser Methode alle Aktionen kennen, die der Lernende während der Lösung der Aufgabe gemacht hat und kann sie dann mit den Schritten vergleichen die er selbst gemacht hat. Diese Methode eignet sich dann wenn es wenige differenzierte gute Lösungswege gibt oder es wichtig ist, dass der Lernende genau die bekannte Lösungsstrategie verwendet. Als Voraussetzung es allerdings es möglich sein, den Lösungsweg in mehrere Schritte zu unterteilen und dass das ITS die Aufgabe selbstständig lösen kann. Um dies zu gewährleisten konzentriert sich deren Domäne auf die Bereiche welche durch stark formalisierte Regeln beschrieben werden können. Implementierungen in der Physik [2000 GERTNER and VAN LEHN][2001 SHELBY, SCHULZE and TREACY], Mathematik [1997 KOEDINGER, ANDERSON and HADLEY][2002 HEFFERNAN and KOEDINGER][1985 ANDERSON, BOYLE and YOST][1998 HEFFERNAN][1990 WERTHEIMER] oder das Programmieren [1993 CORBETT and ANDERSON][1995 CORBETT, ANDERSON and O'BRIEN][1997 CORBETT and BHATNAGAR]. Der Vergleich der Verhaltensweise der Lernenden mit einem vorgefertigten perfekt korrekten Weg hat den Vorteil, dass eine etwaige Implementierung einer Hilfefunktionalität auf der Fähigkeit des ITS' aufbauen kann, ihm den nächsten Schritt zu zeigen. Besonders die Lernenden, welche sich noch nicht viel mit der Materie befasst haben, haben damit einen roten Faden an den sie zurückgeführt werden. Lernende die außergewöhnliche Lösungswege nehmen (auch weil sie sich schon gut mit der Materie auskennen) können falsch erkannt werden.

Eine bestimmte Art der MTT sind die „Cognitive Tutors“. Diese geben direkt eine Rückmeldung ob der eingeschlagene Lösungsweg aus Sicht des ITS korrekt ist. Ihre Simulation des Expertenentscheidungsprozesses basiert auf einem kognitivem Modell. Cognitive Tutors können zum Beispiel regelbasiert sein, ein kognitives Modell, dass nur anhand von vordefinierten Regeln entscheidet.

Constraint-Based-Tutors (CBT)

Die Constraint-Based-Tutors benutzen ein jüngeres Methode, vorgeschlagen 1992 von Stellan Ohlsson [1992 OHLSSON]. Es basiert darauf, das Domänenwissen nur als eine Menge von Überprüfungsbedingungen zu betrachten. Es stellt damit das Domänenwissen komplett anders dar als MTT [2005 KODAGANALLUR, WEITZ and ROSENTHAL], anstatt eine Lösung zu generieren, werden Ergebnisse mittels Bedingungen überprüft. Abhängig davon welche Bedingungen zu dem gegebenen Ergebnis passen wird die Bewertung und Rückmeldung des ITS angepasst. Im Gegensatz zum MTT kann damit nur die komplette Lösung einer Aufgabe eines Lernenden bewertet werden, es gibt keine Rückmeldung des ITS in Lösungsschritten. Bei der Auswertung der Lösung können Fehler erkannt werden, weil bestimmte Bedingungen nicht erfüllt wurden. Das ITS stellt daraufhin gezielt eine Erklärung und weiteres Lehrmaterial in diese Richtung bereit. Es ist damit ergebnisorientiert, unerheblich welche Aktionen der Lernende gemacht hat. Um die Lösung zu erstellen wird nur das Ergebnis betrachtet. Diese Variante wird weniger Domänen eingesetzt als es beim MTT der Fall ist. Dieses Paradigma für ITS ist besonders geeignet für Domänen, bei denen mit der Lösung selbst detaillierte Informationen vorliegen. Als Beispiel ist die Domäne Mathematik sehr schwierig für CBT, wenn die Lösung nur eine simple Zahl ist. Ohne den Lösungsweg zu betrachten ist die falsche Lösung „42“ nicht sehr aussagekräftig, wenn die korrekte Antwort „0“ wäre. Ein Vorteil von CBT basierten ITS ist, dass sie keine Fehlerdatenbanken benötigen, was die Entwicklungszeit verkürzen kann. Auch sind CBT in Domänen einsetzbar deren Domänenwissen selbst fehlerbehaftet sind [2011 MITROVIC]. Es ist für das ITS dann möglich die Auswertung mit einem unvollständigen Satz an Bedingungen durchzuführen. Das erlaubt eine gewisse Flexibilität, die dazu genutzt werden kann, korrekte Lösungen unabhängig von leichten Veränderungen durch Präferenzen des Lernenden zu finden. Eine passende Domäne sind Domänenspezifische Sprachen (DSL) wie zum Beispiel „Structured Query Language“ (SQL). Beispiel-ITSs sind KERMIT [2002 SURAWEERA AND ANTONIJA MITROVIC and MITROVIC] oder SQL-Tutor [1998 MITROVIC]. Es gibt auch ein Autorensystem mit dem Namen ASPIRE [2009 MITROVIC, BRENT and SURAWEERA].

Example-Tracing-Tutors

Der Ansatz des Example-Tracing-Tutors ist auf der Idee aufgebaut, anhand von handgemachten Beispielen zu bewerten und Rückmeldung zu geben. Die Fähigkeiten des ITS steckt in dem aufwändiger erzeugtem Lehrmaterial bei gleichzeitig wesentlich geringerem Aufwand in den automatischen Analyse des Lehrmaterials. Letztendlich ergibt dies einen Vorteil in den Kosten in der Entwicklung eines ITS durch die Implementierung von großen Teilen des Domänenwissens und Pädagogikwissens mittels einfacher Werkzeuge durch fähige Autoren. Die Bedienung der Werkzeuge benötigen wenig Fertigkeiten in der eigentlichen Entwicklung eines ITS, sondern dienen den Autoren mit Fertigkeiten in Didaktik und der Domäne Beispielabläufe zu kreieren. Diese dienen dann als Referenz für korrekte und definiert fehlerhafte Lösungswege. Solch ein Werkzeug würde es zum Beispiel dem Autoren ermöglichen sich in die Rolle eines Lernenden zu versetzen um dann eigenhändig eine Interaktionsfolge nachzuspielen und abzuspeichern. Einer der größten Kostenpunkte in der Entwicklung eines ITS sind die Ausgaben für das Fachpersonal, dass nicht nur Autoren, sondern auch ITS-Entwickler braucht [2016 ALEVEN, MCLAREN and SEWALL]. Bei Example-Tracing-Tutors würde für das Erweitern des Lehrmaterials kein Programmierer

mehr benötigt. Example-Tracing-Tutors unterscheiden sich von MTT durch ihre statisch von Autoren vorher generierten Lehrabläufe. Jede Aktion wurde von einem Autor absichtlich erstellt. MTT hingegen können selbstständig einen Lösungsweg generieren und auch auf Aktionen des Lernenden reagieren, wenn diese nicht von Autoren vorher betrachtet wurde. Die Bewertung und Rückmeldung sind hingegen ähnlich, beide Vergleichen das Verhalten von dem Lernenden mit einem Referenzverhalten. Ein Beispiel für diese Art der ITS sind die eher unbekannte „Stoichiometry“ [2006 MCLAREN, LIM and GAGNON] oder „French Culture Tutor“ [2006 OGAN, ALEVEN and JONES]. Example-Tracing-Tutors werden hauptsächlich von dem „Cognitive Tutor Authoring Tools“-Framework erzeugt, da diese Forschungsgruppe auch die Methode der Example-Tracing-Tutors 2009 erzeugt hat [2009 KOEDINGER, ALEVEN and MCLAREN]. Andere ähnliche Autorensysteme können auch von Autoren ohne fortgeschrittene Programmierkenntnisse entwickelt werden wie ASPIRE [2009 MITROVIC, BRENT and SURAWEERA], ASSISTments [2009 RAZZAQ, PATVARCZKI and ALMEIDA], SimStudent [2015 MATSUDA, COHEN and KOEDINGER] und xPST [2010 KODAVALI, GILBERT and BLESSING].

Machine Learning Based Tutors

Eine durch die immer größer werdenden Rechenkapazitäten und den vielen frei zugänglichen Machine Learning Softwarebibliotheken ist der Trend des maschinellen Lernens. Es hat eine Ähnlichkeiten in der Idee zu den Example-Tracing-Tutors anhand von Beispielen zu lernen, dies wird bei den Machine-Learning-Based-Tutor ITSs einen Schritt weiter gegangen und automatisiert. Input und Output sind hierbei wohl definiert. Ein Großteil der menschlichen Autorenarbeit wird damit durch automatische Maschinenlernverfahren ersetzt, benötigt aber homogenen und vordefinierten Input und Output. Ein Methode die Entscheidungen für die Korrektheit der Eingabe anhand eines bereits trainierten neuralen Netzes trifft. Dieses Netz wird nicht manuell erstellt, sondern eine Trainingsmenge legt Aufgaben und richtige und falsche Ergebnisse fest an der das Netz trainiert wird. Das ist rechen- und zeitaufwändig. Die Gefahr dabei besteht, dass die Trainingsdaten nicht optimal gewählt wurden. In ihnen befinden sich unter Umständen Gemeinsamkeiten die nicht trainiert werden sollten, welche zu Fehlinterpretation bei dem ITS führen kann. Das Verhalten eines Machine Learning Based Tutor ITS kann praktisch nur durch seine Aktionen überprüft werden. Es kann nicht ausgeschlossen werden, dass solch ein ITS sich manchmal unpassend verhält und selbst kleine Änderungen an Netz können schnell sehr aufwändig werden. Sehr komplexe und fehlerbehaftete Vorgänge für die es keine effizienten und guten Lösungsalgorithmen gibt, können manchmal sehr gut mit maschinellem Lernen gelöst werden, weil nach einer Ähnlichkeit zu einem Muster gesucht wird, darunter zählt die natürliche Sprache und Audio- beziehungsweise Bildinterpretation. Genau diese schlecht zu spezifizierenden Domänen könnten für Machine Learning Based Tutors passend sein.

II.3.f. Menschlicher Lehrer versus ITS

Zur Analyse des Ablaufs des Lernens, wird erst das Referenzverhalten für ein ITS genommen: Das Lehren von einem menschlichen Lehrer in einem 1:1 Kontext. Ein einzelner Lehrer für einen einzigen Schüler ist für die Lerneffizienz optimal [2010 WOOLF], es wird deshalb von einer 1:1 Situation ausgegangen (1 Lehrer, 1 Lernender). Dabei wird ein Modell zur Erstellung von ITSs als Grundlage verwendet, welches auf dem Komponentenmodell als klassisches und akzeptiertes Modell für ITSs gilt. Die Klassen an Gedanken für das Verhalten eines Lehrers lässt sich in das Komponentenmodell eines ITS abbilden mit den für Lehrer üblichen Begriffen, siehe Abbildung R-II3f-1.

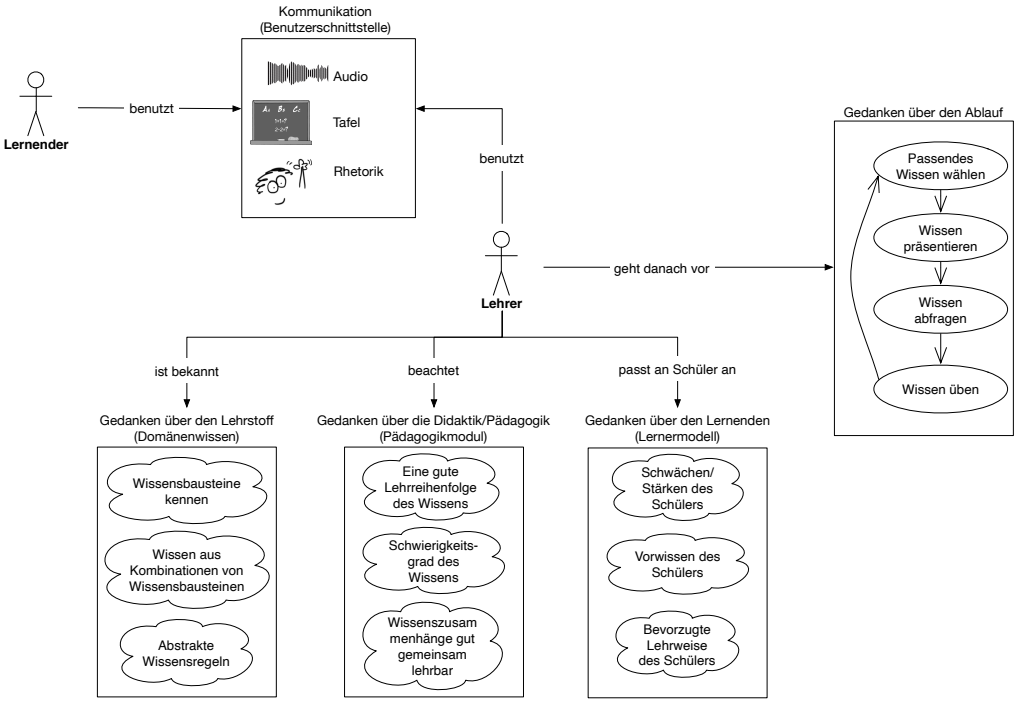


Abbildung R-II3f-1: Komponenten der Verhaltensweisen eines Lehrers im Unterricht

In der Abbildung allerdings fehlt ein wichtiger Teil, die Gedanken über den Ablauf des Lehrens. Ein Teil der nicht in den bisher akzeptierten klassischen Komponenten enthalten ist. Es lassen sich alle bis auf eine Verhaltensweise des Lehrers in die Kategorien aufteilen, welchen der Kernkomponenten bei einem ITS entsprechen. Nun ist ein einfaches Vergleichsmodell aufgestellt an dem man anhand von Komponentenzuordnungen, die man einem Lehrer nach ITS Vorbild zugeteilt hat, was womöglich wichtige Richtungen sind die bisher in der klassischen ITS-Architektur vernachlässigt wurden.

1. Erkennen ob der Lernende aktiv mitarbeitet oder Material verstanden hat
2. Ad hoc neues Lehrmaterial generieren oder vorhandenes variieren
3. Teilen des Wissens zwischen ITSs
4. Lehrmaterial abstrahierter Speichern können
5. Einen Prozess für den Unterricht haben, der keine vorgegebene Sequenz ist
6. Viele Anforderungen an Mindestfunktionalität sind vorhanden die explizit definiert und gefordert werden müssen

Zu 1.: Ein Lehrer kann vor Ort schnell einschätzen, ob ein Lernender aktiv an einer Aufgabe arbeitet, weil er viel mehr seiner Aktionen mitbekommt und diese auch besser einschätzen kann, aber auch nur dann, wenn er auch vor Ort ist. Bei einem Chat zum Beispiel wird das schon erheblich schwieriger. Genau dieser Aufgabe sehen sich auch ITS konfrontiert. Normalerweise setzt ein ITS einfach voraus, dass ein Lernender selbständig aktiv mit dem System arbeitet. Wenn dem Lernenden mit dieser Voraussetzung etwas angezeigt wurde, dann kann davon ausgegangen werden dass er es mitbekommen und sich damit auseinander gesetzt hat. Um später nachzuvollziehen was der Lernende wissen müsste kann man abspeichern was dem Lernenden angezeigt wurde. Doch die Realität ist nicht so simpel. Lernenden wird etwas erklärt und es ist nicht möglich auf Anhieb zu entscheiden ob es verstanden wurde. Bei einer Übung kann das System dann schon erkennen, dass Aufgaben nicht für den Lernenden lösbar sind und der Schwierigkeitsgrad zu hoch war, doch das senkt die Motivation des Lernenden. Es wäre besser möglichst schnell die Fertigkeiten des Lernenden einzuschätzen. Selbst bei Nachfragen des Systems schätzt sich der Lernende unter Umständen falsch ein. Man möchte also dass sich der Lernende bewusst und damit aktiv besonders mit einem für ihn notwendigen und schlecht für das ITS einschätzbaren theoretischen und passiven Teil auseinandersetzt.

Hier gibt es zwei Konzepte wann der Lernende aktiv lernt. Das erste Konzept geht davon aus, dass die häufige Interaktion mit dem System oder den für das Lernen der Domäne relevanten Werkzeugen notwendig ist für effektives Lernen. Dabei ignoriert man den Fakt das ein Lernender das System ausspielen kann. Das zweite Konzept geht davon aus dass Lernen generell in den Gedanken des Lernenden passiert und nicht was der Lernende in der realen Welt macht. Dieser Ansatz verlangt von dem Lernenden ein aktives Verarbeiten des zu lernenden Inhalts. Was der Lernende aktiv denkt ist praktisch unmöglich zu steuern. Das Ziel ist es den Lernenden dazu zu kriegen fokussiert über den Lerninhalt nachzudenken. Selbst wenn die Motivation hoch ist dürfen dabei möglichst wenig ablenkende Faktoren vorhanden sein, auch wenn es Wissen aus der gleichen Domäne ist, sonst wird etwas anderes gelernt als vorgesehen. Das Ziel ist es nicht etwas Wichtiges zu lernen, sondern dass zu lernen, was gerade vorgeschrieben wird, damit möglichst genau nachvollzogen werden kann, was der Lernende über die Domäne weiß. Es sei bemerkt, dass es unproblematischer ist, wenn der Lernende abbricht, weil dann generell davon ausgegangen wird, dass er den Inhalt nicht verstanden hat auch wenn das später zu leichter Unterforderung führen kann. Ein Abbruch wird aber eher als ein Zeichen dafür gesehen, dass wenig Inhalt verstanden worden ist. Ein konkretes Beispiel ist, wenn eine Hilfe- oder Hinweisfunktion zu viele Informationen liefert und damit den Lernenden entweder überfordert oder seinen Fokus gedanklich zu weit von dem eigentlichen Thema wegführt. Ein anderes Beispiel ist, wenn die Motivation im Vordergrund steht und die Benutzeroberfläche und die Benutzerinteraktion (UI/UX) mehr auf die Motivation bedacht ist als auf den Inhalt. Spielerische Elemente welche die Motivation erhöhen, sorgen schnell dafür dass das eigentlich zu lernende theoretische Modell weniger Aufmerksamkeit bekommt. Hierzu zählen jegliche Aufgaben die zusätzlich von dem Lernenden verlangt werden, wie komplizierte Menüführung oder divergierende Bezeichnungen von gleichen Aktionen. Damit dies nicht passiert muss die Funktionalität genau entworfen und umgesetzt werden damit nicht mehr Funktionen angeboten werden als notwendig. Bei einer Einheitlichkeit des zugrundeliegenden Lehrmaterials wäre die Wahrscheinlichkeit von divergierender Benennung geringer. Dies ist ein Problem welches sehr weit in die Psychologie und die UI/UX geht und es muss betrachtet werden in wieweit dies relevant wird bei der Entwicklung der ITS-Architektur. In der Arbeit sollte klar definiert werden, wo die Grenzen und der Fokus in Bezug auf die Beschreibung von UI/UX und Psychologie liegt.

Zu 2.: Der nächste Punkt ist, dass ein Lehrer ad hoc neues Lehrmaterial erzeugen kann. Eine Aufgabe die für ITSs von großen Wert ist. Deshalb sollte das Konzept des Lehrmaterials genauer angeschaut werden, damit ein ITSs zum Beispiel durch eine bestimmte Aufbereitung des Lehrmaterials in der Lage ist Übungsaufgaben zu variieren. In der Doktorarbeit sollte ein Weg für solch eine Generierung gesucht werden.

Zu 3.: Lehrer sind selbst fähig ihr Wissen um neues Material zu erweitern. Autoren als Nutzer mehr Beachtung schenken, weil sie die Fähigkeiten der Wissenserweiterung für das ITS darstellen. Lehrer können sich untereinander austauschen. Eine Möglichkeit des Austauschs zwischen ITSs sollte vorhanden sein. Eine direkte Kommunikation zwischen ITSs wäre zu aufwändig zu fordern, weil dies einen Kommunikationsstandard benötigen würde. Eine zentrale Sammelstelle für Daten von ITSs könnte dazu beitragen, dass Updates der Daten nicht unbedingt Upgrades des Systems bedeuten, besonders dann nicht, wenn die Datenbank zentral verwaltet wird. Dazu müsste das ITS auch nicht einem Standard für die Datenbank folgen, sondern könnte sich gewünschte Teile eines zentralen aktuellen Datensätze in seine Datenbankstruktur transformieren. Deshalb wird die Rolle des Autors für Domänen- oder Pädagogikwissen in der Entwicklung der Konzepte für ITS-Architekturen und in der Architektur selbst präsent sein. In der Arbeit sollte die Rolle des Autors von neuem Wissen untersucht und in die Architektur integriert werden.

Zu 4.: Das Domänenwissen kann schon klassifiziert sein und damit wäre es dem ITS bewusst, welche Wissensbausteine sich ähneln und welches abstrakte Wissen über das

gesamte Domänenwissen existiert. Das Domänenwissen bleibt damit nicht nur eine Ansammlung von für das ITS unverständlichen Daten. Beim Pädagogikwissen geht es nicht um die konkreten Verhaltensweisen und Regeln für jede Übungsaufgabe oder Inhalt im einzelnen, sondern es sind allgemeingültige Regeln der Didaktik vorhanden und es wird direkt grob nach Schwierigkeitsgrad, Ähnlichkeit, ob es für die Zielgruppen passt und was gute Reihenfolgen für die Lehre für verschiedenste Szenarien und Fehlinterpretationen sind kategorisiert. Es ist sollte dem ITS also möglich sein aus dem reinen Domänenwissen mehr zu machen ohne dabei ein mehr an Wissen zu verlangen. Eine Art Metainformationen hinzuzufügen und Zusammenfassungen zu erstellen. Doch es ist nicht einfach diese Erfahrungswerte auszudrücken. Es ist also die Idee es dem menschlichen Lehrer gleich zu tun und das Wissen was schon da ist oder gesammelt werden kann anzureichern durch eine Analyse. Es zeigt sich auch, dass ein Grundprozess eine wichtige Rolle spielt damit allen Beteiligten klar ist, was passiert und damit zeitsparend keine Organisationsdetails erklärt werden müssen. Dieser Prozess muss aber auch flexibel genug sein um auf Lernende einzugehen. Die Kommunikation mit den Lernenden kann konkret unterschiedlich sein ist aber ähnlich zu einer Benutzeroberfläche. An deren Schnittstelle muss der Dialog immer angepasst werden. Es wird festgehalten, dass es bestimmte Stufen der Analyse von vorhandenen Wissen von dem ITS gibt, ein Teil der abstrakt genug ist um wiederverwendet zu werden und ein Teil der vom Wissen über das Lehrmaterial als auch den Lernenden abhängig ist.

Zu 5.: Ein Prozess für ITSs fehlt. Abläufe nach dem ein menschlicher Lehrer den Unterricht gestalten kann gibt es zuhauf. Es wurde schon früh in der Entwicklung dieser Doktorarbeit erkannt, dass solch ein Prozess auch für ITS entwickelt werden muss.

Zu 6.: Die Fähigkeiten eines Lehrers hängen maßgeblich von seiner Kompetenz ab, diese Kompetenz beschreibt aber mehr als das was er weiß, denn dass wäre nur Stufe 1 von Biggs [2001 BIGGS]. ITS sind in Stufe 3. Deshalb sollte auch die Beschreibung was ein ITS ist die Funktionalität dessen widerspiegeln und dass sollte sich auch in der Architektur niederschlagen. Deshalb wird darauf geachtet, dass die Funktionalitäten des ITS entsprechend definiert werden.

Vor- und Nachteile von ITS und menschlichen Lehrern

Allgemeine Vorteile eines ITS rühren teilweise daher, dass es ein digitales Lehrsystem ist. Dieses kann heutzutage auf einem sehr kleinem Mobilgerät wie einem Smartphone oder Tablet laufen und ist damit leicht und immer und überall verfügbar. Besonders in Gegenden und in Situationen, bei denen ein menschlicher Lehrer nicht verfügbar oder für den Einzelnen nicht genügend Aufmerksamkeit bereitstellen kann. Auch wenn man nicht den menschlichen Lehrer ersetzen will, erlaubt es diesem eine gewisse Entlastung in einer Blended-Learning-Umgebung, bei dem die Lehrinteraktion sich zwischen dem menschlichen Lehrer und dem System abwechselt. Ein ITS simuliert einen persönlichen Lehrer. Eine Änderung des Kommunikationsmediums ist teilweise für das ITS problemlos möglich ohne Kosten zu erhöhen, so ist eine Änderung von Text auf gesprochene Ausgabe bei menschlichen Lehrern mit großen Auswirkungen sowohl in der Lerneffizienz, der investierten Zeit pro Lerneinheit als auch für die Lehre überflüssigen Gesprächen verbunden. Bei ITS macht dies kaum ein Unterschied [2006 LITMAN, ROSÉ and FORBES-RILEY]. Ein persönlicher menschlichen Tutor ist besser als ein ITS [2010 FOSSATI, EUGENIO and OHLSSON]. Auch wenn die Lehrergebnisse besser sind, so ist es bei einem Teletutoring manchmal nicht mehr unterscheidbar ob es sich um ein ITS handelt oder eine reale Person, welche auf die Fragen antwortet [2002 PERSON, GRAESSER and TUTORING RESEARCH GROUP].

Die bisherige Forschung im ITS Bereich postuliert folgende Vorteile eines digitalen Lehrers gegenüber einem menschlichen Lehrer.

- Lernende haben einen ITS-Tutor immer für sich alleine
- Ein ITS ist „unendlich geduldig“, besonders bei SchülernLernenden mit einer anderen Muttersprache
- Ein ITS ist immer verfügbar (Ausnahmen sind technische Probleme)
- Ein ITS hat eine geringere Wahrscheinlichkeit Erfolgsdruck auszulösen
- Zwischenmenschliche Ängste sind geringer gegenüber einem ITS (zum Beispiel die Angst vor dem freien Sprechen, die Angst immer wieder zu Fragen oder eine dumme Frage zu stellen)
- Lernende werden nur aufgrund ihrer Leistungen bewertet
- Ein ITS wird nicht beleidigt
- Individuelle Anpassungen der Lerngeschwindigkeit
- Es kann das Problem auf viele Arten gelöst werden, die Lösung kann an den einzelnen Lernenden angepasst werden
- Sofortige Rückmeldung auf Antworten
- Die Aktivität des Lernenden leitet das Lernen
- Zustand eines ITS lässt sich speichern und kopieren
- Ein ITS kann sich immer gleich Verhalten
- Keine Ablenkung durch Dialoge außerhalb des Lehrstoffs (wenn nicht in einem Klassenraumszenario)

Es gibt allerdings auch Nachteile, welche sich nicht von der Hand weisen lassen, darunter fällt folgendes.

- Technisches Verständnis wird von den Schülern erwartet
- Auf Lernende mit außergewöhnlichen oder großen Problemen wird nicht individuell eingegangen
- Externe Motivation nur auf dem Bereich der Interaktivität mit Medien, ein Lerner mit wenig intrinsischer Motivation kann das ITS leicht umgehen
- Betrügereien der Lerner sind nicht so leicht aufzudecken, ein Lehrer kann dies besser Erkennen
- Zwischenmenschliche Gespräche mit dem Lehrer fehlen
- Die Strategien zur Bearbeitung sind auf die technisch umgesetzten Funktionen des vorliegenden ITS beschränkt - ein Lehrer schöpft potentiell aus einem viel größeren, adaptierbaren Fundus

Unterschiedliche Lernende haben verschiedene Bedürfnisse und Vorlieben [2006 CHA, KIM and PARK]. So gibt es das logische Lernen, das visuelle Lernen und das musikalische Lernen [2006 KELLY and TANGNEY]. Die Anpassungsfähigkeit eines menschlichen Lehrers ist dahingehend höher. Es gibt ITS welche ihr pädagogisches Wissen in Form von Lehrstrategien durch die Beobachtung von menschlichen Lehrern erhalten haben [2002 HEFFERNAN and KOEDINGER]. Es ist aber nicht notwendig für ein einzelnes ITS so flexibel zu sein um viele der Bedürfnisse von unterschiedlichen Lernenden abzudecken, weil sich die Vorlieben eines einzelnen Lernenden nicht so rapide ändern. Denn ein ITS unter vielen muss gar nicht mehrere Lernstile abdecken und allen Lernenden gefallen, sondern kann das Lernwerkzeug der Wahl für eine bestimmte Nutzergruppe sein.

II.3.g. Aufteilung des Lehrmaterials

Das Lehrmaterial wurde für diese Doktorarbeit in zwei Kategorien aufgeteilt: interaktives und informatives Lehrmaterial [2019a GRAF VON MALOTKY and MARTENS]. Eine Aufteilung die meistens einfach zu treffen ist, denn der Grad an Aktionsfreiheit bestimmt dessen Kategorie. Das informative Lehrmaterial schränkt die Aktionen des Lernenden ein und das ITS präsentiert und führt, während das interaktive Lehrmaterial vom Lernenden verlangt

selbstständiger eine Aufgabe anzugehen. Dass Lehrmaterial selbst wird bei beiden Kategorien an den Lernenden angepasst. Folgend wird definiert was die beiden Begriffe meinen.

Informatives Lehrmaterial in Lehrsystemen

Bei informativen Lehrmaterial sind die möglichen Aktionen des Lernenden begrenzt. Es werden wenig aktive Handlungen von ihm erwartet. Lehrmaterial wird passiv vom Lernenden aufgenommen, weil erwartet wird, dass dies neues Wissen für den Lernenden darstellt. Die Aktionen des Lernenden sind unter anderem dass Navigieren im Lehrinhalt, die Bestätigung dass der Lehrinhalt passiv zugehört, gesehen, gelesen wurde und dass mehr Erklärungen benötigt werden, weil benötigtes Grundwissen fehlt.

Es dient dazu, den Lernenden eine Einführung in die Materie zu geben, weil er im nächsten Teil vor eine Aufgabe gestellt wird, in dem ihm bewusst sein sollte was er tun kann oder sollte um zur Lösung zur kommen. Das informative Lehrmaterial erwartet hauptsächlich vom Lernenden sich des Kontexts bewusst zu werden, Grundlagen zu verstehen oder nachzuschlagen wenn etwas nicht verstanden wurde. Es verlangt nicht von ihm das Lehrmaterial auch fehlerfrei anwenden zu können. Konkrete Beispiele für informatives Lehrmaterial sind folgend aufgezählt.

- Texte mitunter mit Inhaltsverzeichnis und Verweise zu anderen Textstellen
- Audioaufnahmen mitunter mit Inhaltsverzeichnis und Lesezeichen
- Videos mitunter mit Inhaltsverzeichnis und Lesezeichen
- Statische virtuelle Welt mit nur sehr einfachen Interaktionsmöglichkeiten ohne Fähigkeit des Lernenden zur Veränderung der Welt (zum Beispiel ein Virtuelles Museum)

Interaktives Lehrmaterial in Lehrsystemen

Es ist Material welches dem Lernenden eine Palette an Aktionsmöglichkeiten anbietet und es dann dem diesem überlässt, welche er wählt. Die gewählte Aktion wird vom ITS bewertet, notiert und darauf reagiert. So gibt es direkt eine Rückmeldung vom ITS als auch wird für das zukünftiges Verhalten des ITS die bereits bewerteten Notizen über den Lernenden mit einbezogen. Abhängig vom Lehrsystem können daraus zum Beispiel Rückschlüsse auf das Wissen und Vorwissen des Lernenden gezogen werden. Interaktives Lehrmaterial benötigt eine aktive Interaktion mit dem ITS, das auf Aktionen wartet (abhängig von der Hilfe nur eine gewisse Zeit), die mehr sind als die reine Mitteilung des Lernenden, dass er glaubt es verstanden zu haben. Problemlösungsaufgaben sind ideal um diese Interaktion besser gerichtet umzusetzen. Die Aufgabentypen nach Schanda [1995 SCHANDA] und Mair [2005 MAIR] sind folgend aufgezählt und gehören auch zu den Möglichkeiten des interaktiven Lehrmaterials.

- Einfach-/Mehrfachauswahl, Kreuzauswahl
- Zuordnen
- Reihenfolge bilden
- Einschätzen
- Markieren
- Zahleneingabe, Lückentext, Freie Eingabe
- Minispiele, Spiele, Simulation, virtuelle interaktive Welten ITS-Anwendungsszenarien

Wenn man das Wissen in die Köpfe der Lernenden nachweisbar kopieren könnte, dann bräuchte man nur informatives Lehrmaterial. Wie in der Erklärung zum Wissenstransfer, erklärt in dem Kapitel „Grundlagendefinitionen“, muss es mittels einer Kommunikationsmediums übertragen werden, welches ein Interpretationsspielraum bietet. Es kann durch das interaktive Lehrmaterial eine Verifikation der korrekten Interpretation (Verständnis) und der Erinnerung (Gedächtnis) des informativen Lehrmaterials stattfinden. Um zu testen ob

die Interpretation des Lernenden dem entspricht was erwartet wurde, muss der Lernende anhand seiner Aktionen mit dem interaktiven Lehrmaterial getestet werden. Interaktives Lehrmaterial ist also notwendig für ein ITS um adaptiv und interaktiv zu sein, zusätzlich benötigt man es für ein akkurates Lernermodell um sein Wissenstand zu überprüfen.

II.3.h. ITS-Anwendungsszenarien

Um einen Überblick von den Möglichkeiten der Aufstellung von den einzelnen beteiligten Akteure in den verschiedenen Szenarien zu haben, werden sie einzeln aufgezählt und visualisiert. Das große Ziel ist es zwar, dass das ITS als ganzes einen Lehrer ersetzen kann, in vielen Domänen scheint das unmöglich. Wenn der Lernende Probleme mit dem ITS hat, kann er sich an eine Person wenden, die ihn erklärt was das ITS erwartet oder auch technische Hilfe leisten. Das ITS sollte so konzipiert sein, dass es versucht intuitiv bedienbar zu sein und bei Schwierigkeiten sich selbst zu erklären und erkennen, dass der Lernende Hilfe benötigt. Dies kann Zuviel verlangt sein. Ein Betreuer ist kein Lehrer, denn er hat keinerlei direkte Lehrtätigkeit, sondern stellt einen Überwacher dar, der zusätzlich für eine externe Motivation beim Lernenden sorgt und dabei unterstützen soll, den korrekten Ablauf des Interaktionsprozesses mit dem ITS auszuführen. Sein Ausbildungsstand muss daher nicht so hoch sein wie der eines Lehrers und er muss weder didaktisch noch in der Domäne geschult werden. Das würde Kosten einsparen und es ist in einigen Gegenden der Welt leichter jemanden in einem einzelnen Prozess und dem Umgang mit einem einzelnen ITS neu zu schulen als einen qualifizierten Lehrer zu finden. In armen Gegenden von Afrika kann es günstiger, verfügbarer und effektiver sein lokal keinen Lehrer zu haben und Aufgaben nicht von einem Lehrer gestellt zu bekommen [2015 OLUWATOBI and OLURINOLA]. Einige Schulen haben dort ein System als Hauptquelle für Informationen eingerichtet mit Call-Centern aus Lehrern die jeweils 70 bis 80 Schüler betreuen und das immer nur dann einspringt, wenn inhaltliche Fragen sonst nicht beantwortet wurden.

Die in Gruppen aufgeteilten Szenarios werden folgend einzeln analysiert. Es wird dabei nur auf die Wirkung reduziert welche als Hauptziel von den Lehrakteuren verfolgt wird. Es gibt nur eine begrenzte Anzahl an Akteuren die nachfolgend aufgezählt wird.

- Personalisierter automatisierter Lehrer (Maschine): Ein ITS welches lokal oder über einen Server erreichbar ist und entweder informatives oder interaktives Lehrmaterial anbietet.
- Autor (Mensch): Eine Person welche das Pädagogikwissen oder Domänenwissen über eine Autorenschnittstelle in einem ITS verändert.
- Lehrer (Mensch): Eine Person welche lokal oder über eine technologische Kommunikation erreichbar ist und entweder informatives oder interaktives Lehrmaterial mittels Verwendung eines ITS unterrichtet.
- Betreuer (Mensch): Eine Person welche Lernende begleitet und bei der Interaktion mit dem ITS hilft Unter Umständen kann seine Aufgabe auch die Betreuung der technischen Umsetzung sein.

Es gibt dabei insgesamt fünf grundlegende Wirkungsmethoden: Drei Methoden, wie auf Lernende eingewirkt werden kann und zwei Methoden, wie auf ITS eingewirkt werden kann. Für eine Lehrsitzung ist immer informatives und interaktives Lehrmaterial notwendig, aber es ist nicht erforderlich, dass es beides von dem gleichen Akteur unterrichtet wird. Diese werden folgend aufgezählt.

Wirkung auf Lernende:

- Präsentieren: Informatives Lehrmaterial wird präsentiert, es wird aufgeführt und erklärt mit Inhalten die auf den Lernenden zugeschnitten sind.
- Trainieren: Interaktives Lehrmaterial wird mit dem Lernenden (oder mehreren) durchgearbeitet, es ist ein Dialog zwischen beiden Parteien, Lehrakteuren und Lernendem (Lernenden).

- Betreuen: Die Aufgaben eines Betreuers werden ausgeführt, darunter fallen Motivation und Kontrolle ohne Bewertung des Inhalts.

Wirkung auf ITS:

- Wissensüberarbeitung: Veränderung des Domänen- und Pädagogikwissens vom ITS.
- Aushelfen: Ein Lehrer übernimmt die Aufgabe des ITS für kurze Zeit, weil dieses an seine Grenzen kommt. Das ITS akzeptiert die Einschätzung als korrekt und merkt sich diese eigene Schwäche

Die letzte Aufteilung ist, ob die an der Lehre beteiligten Akteure lokal beim Lernenden sind oder nur mittels Kommunikationstechnik agieren. Technologien für die Fernlehre sind zum Beispiel Videotelefonie oder ein spezielles Programm welches direkt die Interaktionen von Lernenden einsehen und überwachen oder auch einschränken kann. Die beiden Optionen wären dann folgend aufgezählt.

- Lokal: Akteur ist lokal bei dem Lernenden (den Lernenden).
- Distanziert: Akteur ist an einem anderen Ort und über eine technologische Kommunikationsschnittstelle mit dem Lernenden (den Lernenden) verbunden um nahezu sofort eine Rückmeldung zu geben.

Es wird davon ausgegangen, dass ein Lehrer immer ein besserer Betreuer ist als es ein Betreuer sein kann, weil er sich mit dem Domänenwissen auskennt und unterrichten kann. Deshalb wird kein Betreuer benötigt, wenn ein Lehrer vor Ort (Lokal) ist und kein distanzierter Betreuer gebraucht, wenn es einen distanzierten Lehrer gibt. Es wird auch davon ausgegangen, dass ein Lehrer immer besser unterrichtet als ein ITS. Es wird davon ausgegangen, dass die Lehrfunktionalität eines ITS nicht aufgeteilt wird auf zwei Systeme, weil dies kostenineffizient wäre.

Szenariengruppe: Lernender und ITS

In dieser Gruppe von Szenarien gibt es genau einen Lernenden und ein ITS. Wie in Abbildung R-II3h-1 zu sehen entstehen zwei verschiedene Szenarien. Das ITS kann lokal (1) oder über eine Distanz (2) agieren. Lokale ITS wären zum Beispiel ITS-Software auf PCs in der jeweiligen Institution oder Apps auf Smartphones oder Tablets, die ohne eine Internetverbindung funktionieren können. Über die Distanz wäre das ITS zum Beispiel ein Server der durch einen Klienten angesprochen wird, der häufig eine Oberfläche eines Webbrowsers hat, aber auch durch Apps und Programme realisiert werden kann.

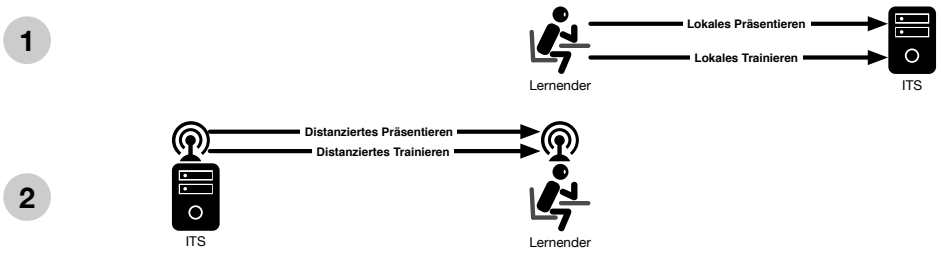


Abbildung R-II3h-1: Nur ITS und Lernender

Lehraufgaben werden aufgeteilt

In dieser Gruppe von Szenarien gibt es zusätzlich zum ITS auch noch einen Lehrer, der entweder das interaktive Lehrmaterial abnimmt und unterrichtet oder das informative. Dies kann entweder komplett lokal wie in Abbildung R-II3h-2, komplett distanziert wie in Abbildung R-II3h-3 oder gemischt wie in Abbildung R-II3h-4 vonstatten gehen. Komplett lokal wie in Szenario (3) und (4) erlaubt es das ITS ohne zusätzliche Verbindung zum Internet zu betreiben, verlegt aber den Wartungsaufwand auch auf den Lehrort. Mit anwesendem Lehrer könnte dieser bei Problemen oder Ausfall des ITS notfalls einspringen. Das ITS ermöglicht es den Lehrer zu entlasten, indem er nur einen Teil der Lehre über-

nimmt und somit mehr Lernende parallel oder sequenziell (zum Beispiel zwei Gruppen zeitlich verschoben) unterrichten kann.

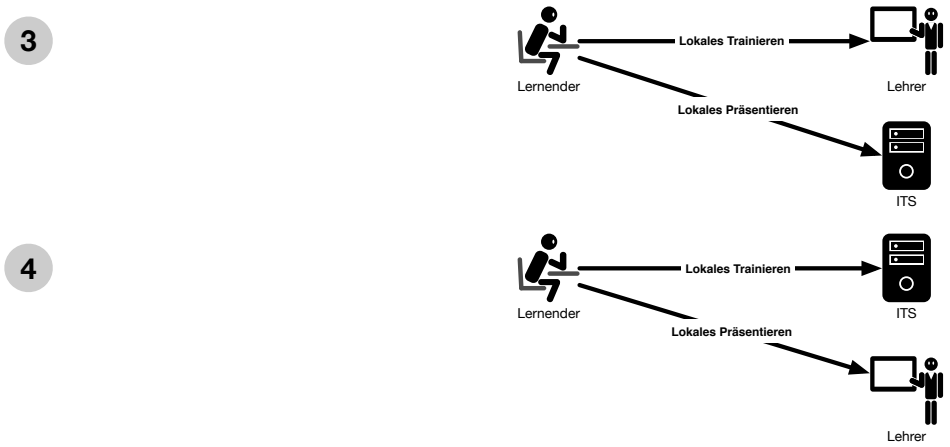


Abbildung R-II3h-2: Lokale Aufteilung der Lehre

Komplett distanziert wie in Abbildung R-II3h-3 mit Szenario (5) und (6) erlaubt es dem Lernenden ortsunabhängig von überall aus unterrichtet zu werden und das zum einen Teil von einem Lehrer und zum anderen Teil von einem ITS.

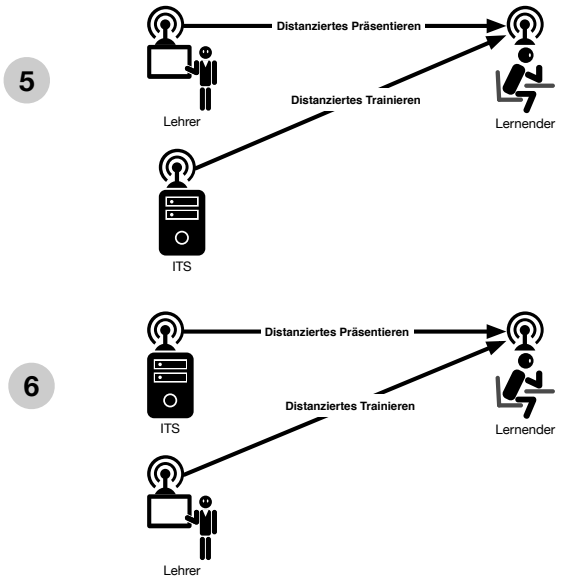


Abbildung R-II3h-3: Distanzierte Aufteilung der Lehre

Die Aufteilung der Lehre zwischen einem ITS und dem Lehrer kann auch in einem Teil lokal und in einem anderen distanziert sein, wie in Abbildung R-II3h-4 zu sehen. Einerseits ist dann ein Lehrer anwesend wie in Szenario (7) und (8), andererseits kann es auch so gestaltet werden, dass das ITS lokal verfügbar ist, wie in Szenario (9) und (10). In beiden Fällen würde das ITS entfernt verwaltet und der Lehrer hätte weniger administrativen Aufwand. Wie bei Szenario (3) und (4) könnte der Lehrer unter Umständen mehr Lernende unterrichten, weil das ITS ihm einen Teil des Unterrichts abnimmt.

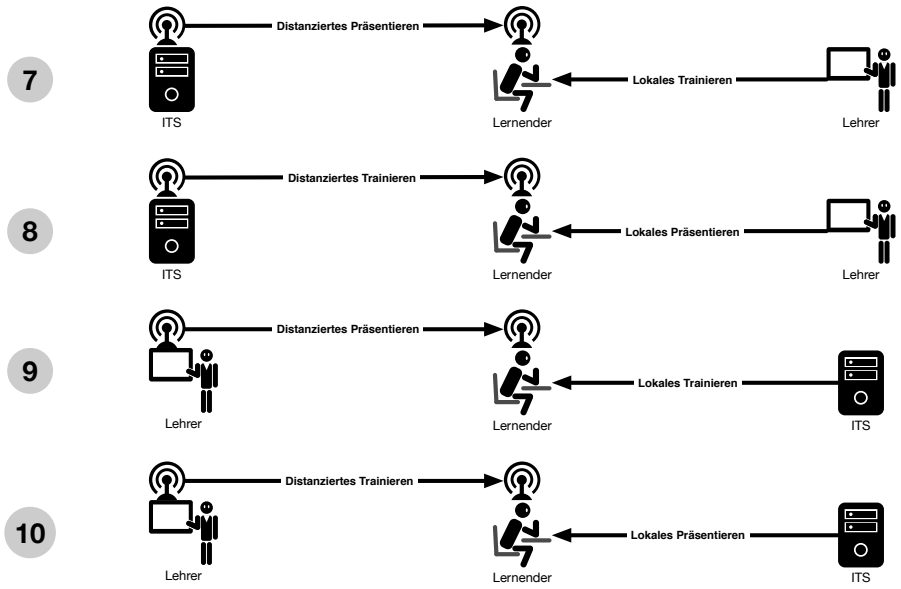


Abbildung R-II3h-4: Distanziert und lokale Aufteilung der Lehre

Lokaler Betreuer integriert

Fügen wir den bekannten Lehrakteuren noch den Betreuer hinzu ergibt das viele neue Szenarien. In dieser Gruppe von Szenarien von (11) bis (16) gibt es einen lokalen Betreuer. Einmal mit einem lokalen ITS in den Szenarien (11) bis (13) und dann mit einem distanziertem ITS in Szenario (14) bis (16), siehe hierzu Abbildung R-II3h-5. Ein lokaler Betreuer gibt dem Lernenden den sozialen Kontakt und die Sicherheit persönlich mit einem Menschen reden zu können. Zudem hat ein Mensch in einer Aufsichtsrolle die Aufgabe dafür zu sorgen, dass Lernende nicht andere Dinge machen. In Szenario (11) übernimmt das ITS die Lehraufgaben und der Betreuer die sozialen Aufgaben von einem Lehrer. In Szenario (12) und (13) gibt es zusätzlich noch einen distanzierten Lehrer, der einen Teil der Lehraufgaben von dem ITS abnimmt und damit mehr Lernende unterrichten kann, als würde er alle Lehraufgaben übernehmen. In dem einen Fall übernimmt der Lehrer nur das interaktive Lehrmaterial, dies in Szenario (12) und in dem anderen Fall übernimmt der Lehrer nur das informative Lehrmaterial, siehe Szenario (13).

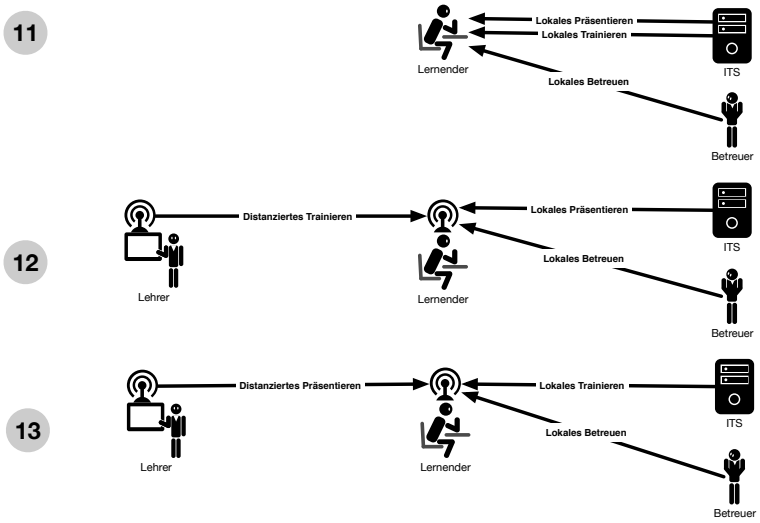


Abbildung R-II3h-5: Lokales ITS mit lokalem Betreuer

In Abbildung R-II3h-6 sind die Szenarien (14), (15) und (16) ausgeführt bei denen das ITS distanziert ist. Diese Szenarien sind sehr ähnlich zu den drei vorherigen, also (11), (12) und (13). Das ITS muss nicht von dem Ort verwaltet werden . Der Betreuer soll und kann

dann in den Szenarien (14) bis (16) sich nicht um die Verwaltung des ITS kümmern, es muss sich darauf verlassen werden, dass die Verbindung zum distanzierten ITS klappt. Vorteil bei solch einer Methode ist die zentrale Anlaufstelle für die Sammlung und Auswertung von des Lernendenwissens.

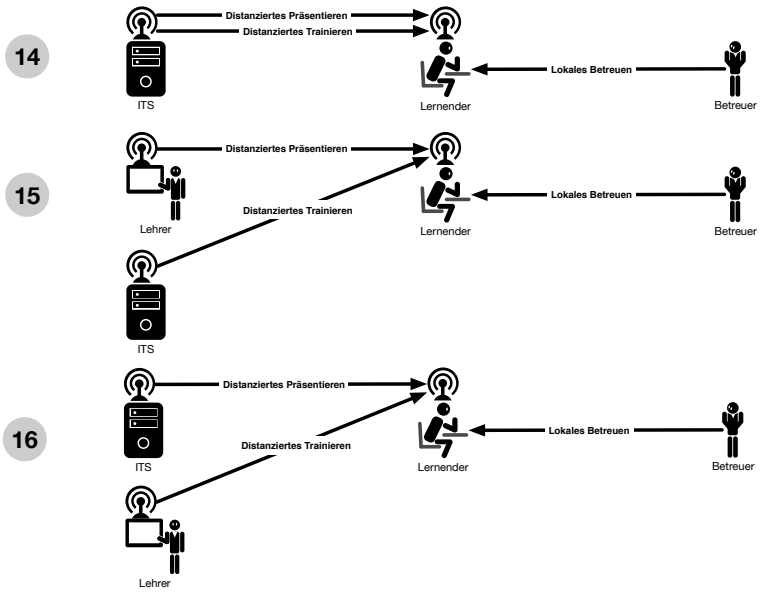


Abbildung R-II3h-6: Distanziertes ITS mit lokalem Betreuer

Distanzierter Betreuer integriert

Der distanzierte Betreuer ist eine weitere Gruppe von Szenarien (17) bis (19), siehe Abbildung R-II3h-7, die sehr ähnlich ist zu der vorherigen Gruppe von Szenarien (11) bis (16). Der Unterschied ist jetzt, dass der Betreuer nicht mehr lokal beim Lernenden ist sondern nur noch über andere Technologien erreichbar ist. Dabei ist der Effekt eines distanzierten Betreuers auf die Motivation wahrscheinlich schwächer. Im Szenario (17) könnte ein Betreuer als Hilfestellung für die Benutzung des ITS und dem zugrundeliegenden Prozess eine wichtige Anlaufstelle für den Lernenden sein, um überhaupt den Einstieg zu finden mit einem ITS zu arbeiten. Der Lehrer in Szenario (18) und (19) muss sich mit einem vorhandenen Betreuer weniger um ITS-Angelegenheiten kümmern und kann sich ganz auf seinen Teil der Lehre konzentrieren.

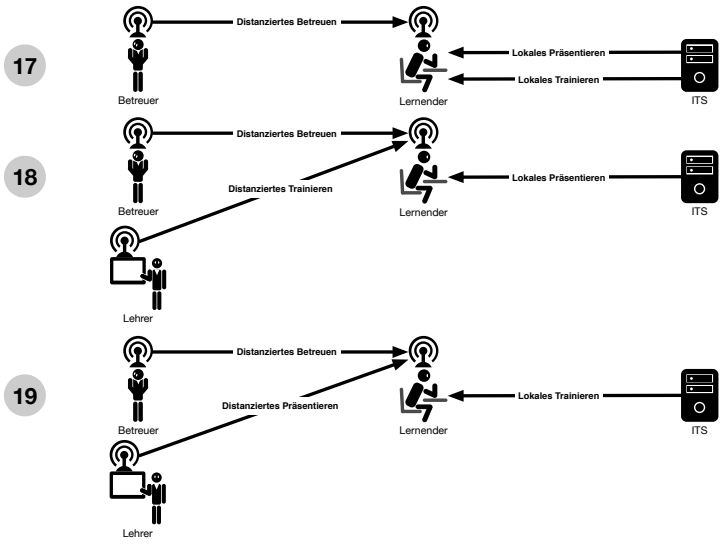


Abbildung R-II3h-7: Lokales ITS mit distanziertem Betreuer

In den folgenden Szenarien (20) bis (22) wird das ITS nun auch distanziert, damit agiert der Lernende völlig losgelöst von lokalen Einschränkungen, bis auf seine Verbindung zu den Lehrakteuren, siehe Abbildung R-II3h-8. Der Lernende muss sich nicht darum kümmern etwas zu installieren oder irgendwohin zu gehen, dies spart Zeit und erlaubt es auch Lernenden unterrichtet zu werden, die nur zu ungünstigen Zeiten oder wenig Zeit am Tag dafür zur Verfügung haben. Um sich nicht komplett allein gelassen zu fühlen ist der Betreuer womöglich noch wichtiger als in den Szenarien (17) bis (19).

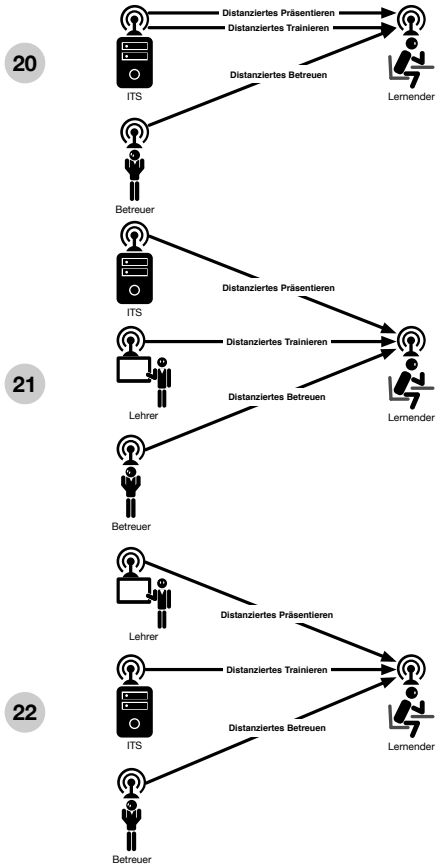


Abbildung R-II3h-8: Distanziertes ITS mit distanziertem Betreuer

Lehrer hilft aus

Die Idee, dass ein ITS komplett die Lehre übernimmt wird nicht immer funktionieren. Selbst wenn nur ein kleiner Anteil der Interaktionen mit dem Lernenden dem ITS Probleme bereitet, so kann die Unfähigkeit eines ITS in einem, wenn auch sehr speziellen Fall, einen Lernenden sehr frustrieren und seine Motivation als auch das Vertrauen in das ITS verringern. Anstatt das eine Lehrer für solch einen Fall diese Lehre zu übernehmen muss, kann dieser im Hintergrund agieren. Der Lehrer übernimmt dabei den Lehrauftrag des ITS nur für den kurzen Moment, in dem das ITS unfähig ist. Dies darf nicht häufig passieren, sonst könnte der Lehrer auch gleich selbst unterrichten. Der Lehrer kann so für eine sehr große Anzahl an Lernenden immer nur sehr kurz zur Verfügung stehen, um dem ITS auszuhelfen. Um zu verhindern, dass die Belastung des Lehrers nicht zu hoch wird, weil die Lernenden sich vielleicht nur noch direkt an den Lehrer wenden würden, dürfte der Lehrer nicht direkt, sondern nur indirekt über das ITS interagieren. Das ITS müsste allerdings die Möglichkeit bieten fremdgesteuert zu werden. Da das ITS sowieso durch digitale Wege fremdgesteuert wird ist es auch nicht notwendig, dass der Lehrer lokal da sein muss. Es ergeben sich die Szenarien (24) bis (26) mit einem lokalem ITS und Szenarien (27) bis (30) für ein distanziertes ITS. Mit lokalem ITS der Szenarien (24) bis (26) wie in Abbildung R-II3h-9 zu sehen muss das ITS eine Verbindung zum Lehrer aufbauen können, also ist das

ITS nicht so unabhängig wie der bisherige Lehrakteur eines ITS. Der Lehrer ist hier kein direkter Ansprechpartner für den Lernenden, sondern nur eine Hilfe für ITS. Deshalb ist es vorteilhaft einen Betreuer zuhaben der entweder distanziert wie in Szenario (25) oder lokal wie in Szenario (26) dem Lernenden beiseite steht. Ein lokaler Betreuer kann mehr Einfluss auf den Lernenden ausüben, während ein distanzierter Betreuer womöglich einfacher zu den Lernzeiten des Lernenden passt.

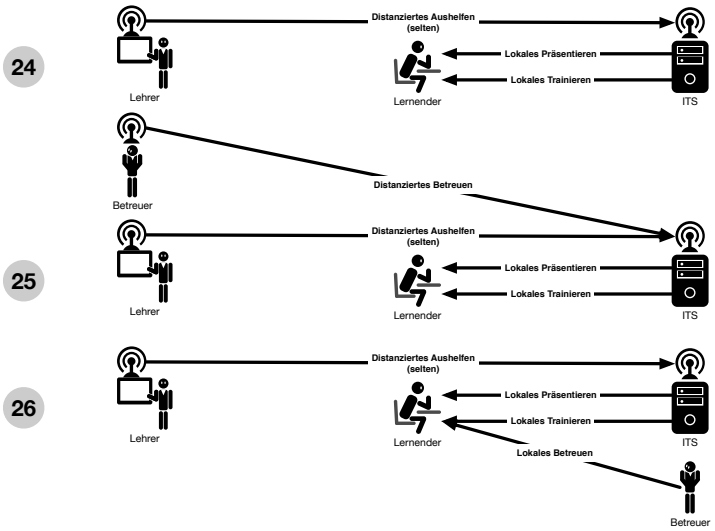


Abbildung R-II3h-9: Distanziertes aushelfen mit lokalem ITS

Nun ist auch auch möglich das ITS zu distanzieren, wie in Szenario (27) bis (29), siehe Abbildung R-II3h-10. Das ist für den Lernenden unter Umständen kaum einen Unterschied von Szenario (24) gegenüber Szenario (26), außer dass eine Verbindung zum ITS aufrecht gehalten werden muss. Dies hängt aber auch von der Implementierung ab. Szenario (27) und (28) beinhalten auch hier einen Betreuer, einmal distanziert in Szenario (27) und einmal lokal in Szenario (28). Dieser ist besonders wichtig, weil es sonst zu wenig Bezug zu dem Lernstoff gibt. Es ist ein distanziertes ITS welches womöglich über den Browser aufgerufen wird und der Lehrer agiert nur verdeckt. Ohne Betreuer muss der Lernende sich selbstständig dazu ermutigen weiter zu machen, denn es ist nicht mal etwas installiert. Das könnte allerdings auch die Hürde zum testen eines ITS reduzieren, da im Notfall der Lehrer vorhanden ist für das ITS und es kaum Aufwand ist zu beginnen.

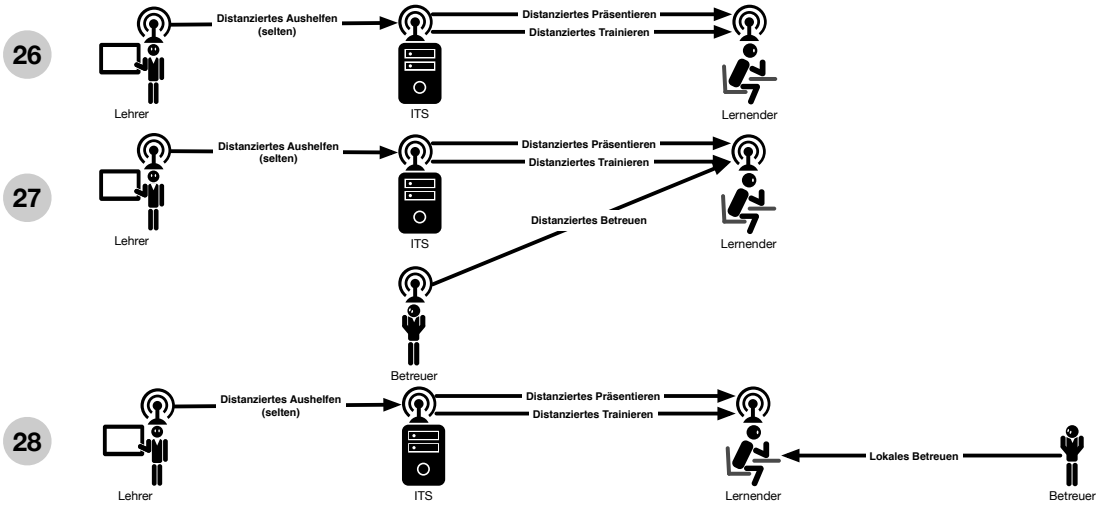


Abbildung R-II3h-10: Distanziertes aushelfen mit distanziertem ITS

Author bearbeitet ITS-Wissen

Die letzte Gruppe der Szenarien ist anders, weil das Ziel nicht die Lehre von Lernenden ist, sondern des ITS selbst, siehe Abbildung R-II3h-11. Deshalb sind es auch die einzigen beiden Szenarien in denen der Lernende fehlt. Hierbei ist es günstig wenn das ITS eine Benutzeroberfläche für Autoren vorweisen kann, damit es einem Domänenexperten ohne tiefgreifendes Verständnis des ITS möglich ist Pädagogikwissen oder Dömanenwissen zu editieren. Das umfasst das hinzufügen von neuen Lehrmaterial und neuem didaktischem Verhalten. Konkret kann das heißen: Regeln für das Verständnis von Domänenwissen zu schreiben, sequenzialisieren von Lehraufgaben, Topologie von Begriffen, bestimmten von Schwierigkeitsgraden, Datenbank über häufig gemachten Lernendenfehlern erweitern, Verknüpfen von Lehrmaterial, Hilfestellungen zu definieren oder Klassifizieren von Lehrmaterial. Quasi all das woraus das ITS Material bereit stellt und dahingehend bewertet, ohne dabei den Lernenden zu betrachten. Der Autor kann ein Lehrer sein, könnte aber auch ein spezialisierter so genannter „Knowledge Engineer“ sein.

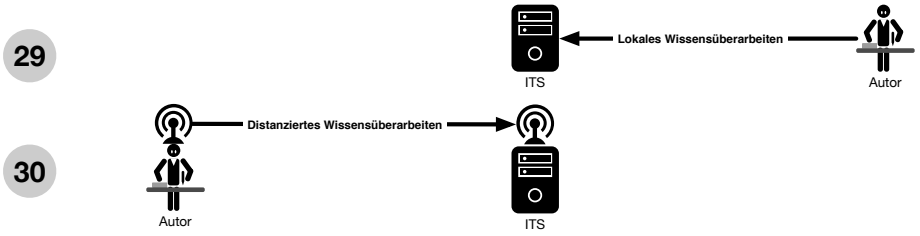


Abbildung R-II3h-11: Autorenszenarien

Erweiterung der Szenarien

Betrachtet man nicht nur einen einzelnen Lernenden, sondern auch eine Gruppe als einen Akteur, dann gibt es zwei Optionen, siehe Abbildung R-II3h-12. Erstens, eine kollaborativ arbeitende Gruppe von Lernenden, welche gemeinsam die Lösung erarbeiten können und zweitens eine Gruppe von isoliert arbeitenden Lernenden. Diese Optionen vervielfachen die Szenarien, denn alle vorgestellten Szenarien die den Lernenden enthalten können jetzt in drei verschiedenen Variationen vorkommen, es entstehen dann insgesamt 86 Szenarien (86=28*3+2). Das Arbeiten von mehreren isolierten Lernenden würde zwar nicht die Aufgaben des einzelnen Lernenden verändern, da es noch immer auf einen Lernenden ausgerichtet ist, aber womöglich kann das ITS Rückschlüsse von der gesamten Gruppe ziehen. Die Lernenden in dieser Gruppe werden wahrscheinlich ähnliche Ansprüche und Fähigkeiten besitzen. Als Beispiel könnten Fehler, die ein Lernender macht auch Problemfelder für die anderen Lernenden in der Gruppe sein, weil sie gleiches Vorwissen haben. So wäre ein ITS in der Lage eine gruppenweite Anpassung vorzunehmen. Eine gruppenweite Anpassung hätte wiederum Vorteile für den Einzelnen in der Gruppe, falls die Gruppe relativ homogen ist. Komplett anders sieht es bei den kollaborativ arbeitenden Lernenden aus, diese haben ganz andere Anforderungen an das ITS [2001 JERMANN, SOLLER and MUEHLENBROCK]. Das ITS wäre nicht mehr einziger Ansprechpartner für die Lernenden und es muss verhindert werden dass sich Gruppendynamiken entwickeln, welche das Lernen (auch nur für Teile der Gruppe) unterbinden. Es ist sehr schwierig das Lernermodell für einzelne in der Gruppe zu erfassen. Tiefergehend wird in dieser Arbeit nicht die Umsetzung von kollaborativ arbeitenden Lernenden analysiert, es ist ein wichtiger Aspekt der optional in einem Konzept vorhanden sein muss. Die Option, die Anzahl der Betreuer oder Lehrer zu erhöhen sind zwar theoretisch möglich aber ineffizient, da es gerade beim ITS auch darum geht Personal verringern zu können.

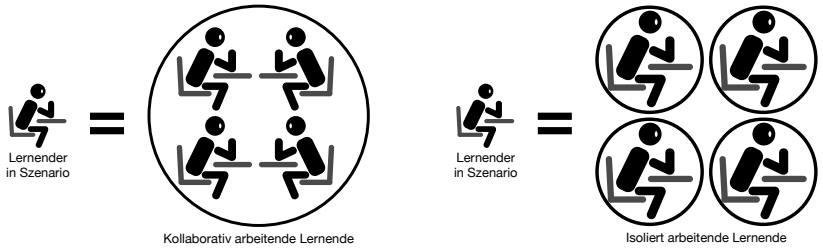


Abbildung R-II3h-12: Optionen für das Ersetzen eines Lernenden in den genannten Szenarien mit einer Gruppe

II.3.i. ITS-Anwendungsfälle

Um einen möglichst hohen Nutzen von einem ITS zu bekommen sollte es wohl bedacht in den bestehenden Lehrablauf integriert werden. Spezielle Forschungen untersuchen Analysen, ob ein ITS heutzutage sinnvollen Einsatz findet, zum Beispiel als Hausaufgabe und Klausurvorbereitung [2009 JOHNSON, PHILLIPS and CHASE][2009 MENDICINO, RAZZAQ and HEFFERNAN][2014 FENG, ROSCHELLE and HEFFERNAN][2019 MITROVIC]. Herausragend bei der Arbeit von [2019 MITROVIC] ist dabei, dass selbst bei komplett optionaler Verwendung eines ITS, in diesem speziellen Fall der EER-Tutor, die Studenten zu 75% im Jahr 2018 diesen genutzt haben und bei einer Lösung von mindestens 5 der vorhandenen 16 Übungen, sie wohl in den ITS unabhängigen Hausaufgaben und der Klausur deutlich besser abschnitten. In einer großen Studie (50 kontrollierte Einsätze) waren 92% der Lernenden besser, wenn sie statt des konventionellen Unterrichts, ein ITS als Unterrichtswerkzeug bekamen [2016 KULIK and FLETCHER]. In welchen Fällen der Einsatzzweck aktuell am effizientesten ist muss noch weiter untersucht werden. Wenn das ITS versucht einen menschlichen Lehrer zu imitieren ohne seine eigenen Vorteile auszuspielen, stehen diese in Konkurrenz. Das ITS kann die Qualität eines Lehrers aktuell nicht erreichen. ITS ist ein Lehrwerkzeug und aktuell nicht der vorrangige Weg zum lernen. Ebenso mag der Unterricht mit einem menschlichen Lehrer, als der Weg mit dem höchsten Wissenszuwachs, nicht zwangsläufig die beste Wahl für jede erdenkliche Situation sein.

Im vorherigen Abschnitt wurden schon die verschiedenen Szenarien vorgestellt. Zwei der wahrscheinlich effektivsten Szenarien wurden herausgesucht und werden folgend als Anwendungsszenarien aufgeführt. Es ist noch nicht richtig klar, welche Szenarien sich im Endeffekt durchsetzen werden. Es hängt von den Nutzern ab, die diese einsetzen. So wird ein ITS heutzutage wohl eher in Industrieländern verwendet bei denen Technologie weit verbreitet in den Haushalten und Bildungseinrichtungen vorhanden ist. Doch in Zukunft könnte eine drastische Verbreitung des Smartphones in bildungsarmen Regionen der Welt, andere Szenarien bevorteilen, als Beispiel sei die Revolution von Services (Bildung durch „ENEZA“ [2015 OLUWATOBI and OLURINOLA], Banking durch M-Pesa [2011 WILLIAM and SURI]) durch herkömmliche Mobiltelefone in Südafrika genannt.

Anwendungsfall: Klassenraum und zentral verwaltetem ITS mit Videochat-Betreuer

Ein Beispielszenario ist eine Schulklasse, welche örtlich getrennt von dem Betreuer und den Lehrmaterialien ist, ein Fall von „distance education“. Dies kann bei stetigem Fachkräftemangel und schnell veraltenden Domänenwissen (zum Beispiel für eine Fachkraftausbildung im Softwarebereich) von Vorteil sein. Somit ist es möglich die Lehrmaterialen zentral zu verwalten und aktuell zu halten. Kein Lehrer wird benötigt und der Betreuer muss nicht zum Unterrichtsort kommen. Dies verringert die Kosten. Zudem benötigt die Klasse nur einfache Technik, wie zum Beispiel einen aktuellen Browser, denn das eigentliche Lehrsystem für die Berechnungen des ITS und die dazugehören Sammlung an Lehrmaterialen befindet sich nicht lokal bei den Lernenden, siehe dazu Abbildung R-II3i-1.

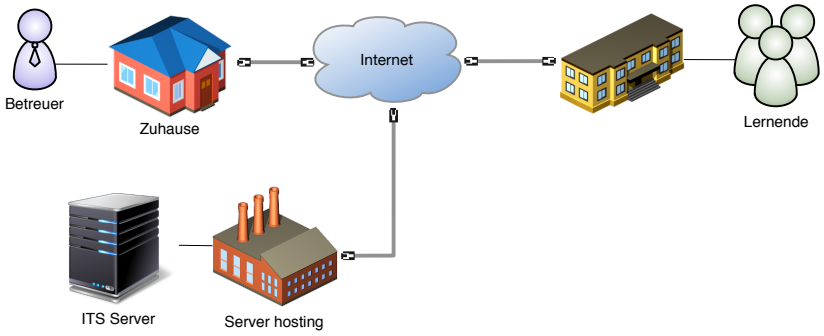


Abbildung R-II3i-1: Örtlich versetztes Lernen

Ein Ablauf wäre, dass eine Schulklasse in das Computerlabor geht um eine Lehrstunde mit dem Computersystem zu bekommen. Bei Schulen mit zu wenig Lehrern würde dies eine Entlastung bedeuten. Die Computer verbinden sich mit einem zentral verwaltetem ITS Server auf dem auch Lehrmaterialen verwaltet werden. Falls die Lerner Hilfe mit dem Umgang des ITS benötigen, ist diese in Form eines Live-Chats mit dem Betreuer möglich. Dieser kann zum Beispiel auch mehrere Klassen gleichzeitig beaufsichtigen. Das wäre eine kostensparende Möglichkeit Unterricht zu gestalten. Erfordernisse sind hierbei eine zum Zeitpunkt der Unterrichts ständig aktive Internetverbindung und Schüler die Selbstkontrolle beherrschen, zum Beispiel Gymnasiasten oder Weiterbildungen von Erwachsenen.

Anwendungsfall: Flexibles Selbstlernen über Mobilgeräte

Ein zweiter Anwendungsfall wäre, dass Lernende mit Hilfe einem ITS auf einem mobilen Endgerät sich Domänenwissen aneignen ohne etwas anderes zu benötigen. Mehr als ein Gerät sollte dafür dann nicht benötigt werden, siehe Abbildung R-II3i-2. Dieser Trend wurde schon 1998 erkannt [1998 LELOUCHE] und ist durch die Smartphones und den Blended-Learning-Stil stark ausgeweitet worden. Viele heutige erfolgreiche Lehrprogramme basieren auf diesem Prinzip. Die Lehrmaterialien befinden sich lokal auf dem Gerät und können durch Updates auch aktualisiert werden. Das ITS geht hierbei nur von einem Lernenden aus. Eine Beispielsanwendung wäre das mobile Lernen mit dem ITS-Programm, wenn es zeitlich und/oder örtlich nicht machbar wäre einen Unterricht mit einem Lehrer zu ermöglichen. Es ist sogar möglich, dass das ITS sich geräteunabhängig auf einem austauschbaren Datenträger befindet und mobil auf jedem zur Verfügung stehenden Computer verwendet werden kann. Eine andere Anwendung dieses Falls wäre, dass es eine Übungsphase für Zuhause gibt, bei dem die Lernenden vor oder nach dem klassischen Unterricht mit dem ITS lernen.

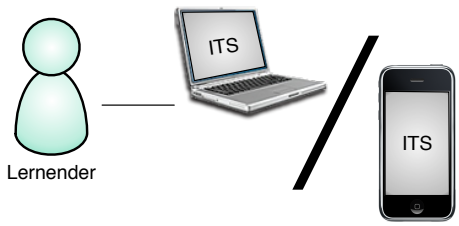


Abbildung R-II3i-2: Mobiles lernen

Das ITS kann damit einen persönlichen Ausbilder darstellen, welcher angepasst an die Anforderungen des Lernenden zu jeder Tages- und Nachtzeit zur Verfügung steht. Anpassungen enthalten unter anderem die Lerngeschwindigkeit, die Stärken und Schwächen in einzelnen Teilgebieten und das Vorwissen. Die Auswertung kann live von dem jeweiligen Interessierten über alle Interaktionen der Lerner, welche mit diesem System arbeiten angezeigt, synchronisiert und ausgewertet werden. Somit kann der Lehrer auch Rückmel-

dung darüber bekommen worin der einzelne Lernende unterstützt werden muss, sofern der Lernende seinen Daten verschickt. Damit die Lehrstrategie an die Lerngruppe oder wissenschaftliche Fakten über das Lernen insgesamt zu erfassen ist schwierig, weil es eine direkte Interaktion benötigt. Auch wäre der Verwaltungsaufwand bei der Lehrinstitution, welche die Geräte ausgibt oder dem Lernenden, selbst wenn es sein eigenes Gerät ist, hoch.

II.3.j. Zustand eines ITS

Ein ITS muss als Grundlage für alle Aufgaben Daten speichern können. Bisher gibt es drei Wissensdatenbanken, welche von dem klassischen Modell übernommen werden können: Domänenwissen, Pädagogikwissen und Wissen über den Lernenden. Auch wenn die Modellierung nicht zu konkret werden darf um allgemein gültig zu bleiben, so lässt sie abstrakt noch eine weitere Modellierung zu. Bei der Betrachtung des Lehrers ist aufgefallen, dass der Ablauf durch einen Prozess grob vorgegeben ist und aktuell in den klassischen Komponenten fehlt. Um diesem Prozess zu folgen wird nicht nur ein Langzeitgedächnis benötigt, sondern auch eines, dass nur relevant für die aktuellen Interaktionen mit dem Lernenden ist. Eine Zustandsbeschreibung für ein ITS könnte dabei die exakte Beschreibung für ein ITS liefern, die für einen Prozess benötigt werden kann. Damit kann man nachträglich das Verhalten von ITS besser analysieren oder das ITS direkt in einer Situation wiederherstellen. Für eine Software teilt sich damit der Zustand in zwei Teile die nachfolgend beschrieben sind.

Persistenter Zustand

Es gibt den persistenten Zustand. Dieser stellt alle Daten dar, welche persistent gespeichert werden, welche auch nach dem Beenden eines ITS erhalten bleibt. Dieser muss zum Beispiel in Dateien in einem Dateisystem gespeichert werden. Es muss davon ausgegangen werden, dass die Datenbank immer weiter wachsenden wird, weil das Lernendenwissen bei Benutzung von Lernenden immer weiter wächst und Autoren das Pädagogikwissen und Domänenwissen erweitern können. Eine Möglichkeit zur effizienten Implementierung wäre eine relationale Datenbank.

Temporärer Zustand

Zusätzlich zum persistenten Zustand gibt es auch Daten die nur zur Laufzeit benötigt werden. Diese zwischengespeicherten Daten werden nach Abschließen der Lehreinheit oder sogar schon nach Schließen einer Ansicht, spätestens aber nach Beenden des ITS nicht mehr benötigt. Deshalb muss auch kein Zugriff zu einem späteren Zeitpunkt mehr möglich sein und von vornherein gar nicht erst persistent und damit langsam gespeichert werden. Diese Daten sind nicht wichtig genug, als dass sie bei einem Fehler wiederhergestellt werden müssen, man generiert einfach neue. Bei einer steigenden Betriebsdauer eines ITS wird der benötigte Speicherplatz des Temporären Zustands stabil bleiben. Beim Ausschalten des ITS sollte der Temporäre Zustand gelöscht werden. Der temporäre Zustand sollte somit bei nicht aktivem ITS leer sein. Wichtige Informationen die davon erhalten werden sollen, müssen damit immer zum persistenten Zustand transformiert werden.

Gesamtzustand

Der Gesamtzustand eines ITS besteht aus persistentem Zustand und temporärem Zustand. Wenn sich eines von beiden ändert, dann ändert sich auch der Gesamtzustand.

II.3.k. Automatische Generierung von Lehrmaterial

Das übergeordnete Ziel eines ITS ist der Wissenstransfer von Domänenwissen zu dem Lernenden. Hierzu braucht das ITS Pädagogikwissen. Doch alleine das Domänenwissen zu kennen reicht nicht aus um gut zu lehren, wie aufgezeigt ist das Pädagogikwissen ein integraler Bestandteil für die Lehre und hat wie das Domänenwissen seine eigene Komponente in der klassischen ITS-Architektur. Es ist hierbei wichtig Pädagogikwissen vom Domänenwissen (zum Beispiel für die Wiederverwendbarkeit) klar zu trennen. Dies ist wie bereits weiter oben aufgezählt eines der häufigsten Fehler von ITSs. Um diese Trennung klar zu machen muss man zuerst Wissen was Domänenwissen ist und sich daran halten ohne mehr als definiert hinein zu interpretieren. Überraschenderweise sind deren Definitionen für ITS nur kurze allgemein verständliche Definitionen. Ganz im Sinne des Anspruchs dieser Arbeit wird versucht Domänenwissen und Pädagogikwissen genauer zu definieren, damit diese nicht gemischt werden.

Domänenwissen ist das Wissen von den Domänenexperten was der Lernende im Endeffekt nach Lehre auch wissen sollte. Die Qualität eines Lehrsystems wird häufig ausschließlich an dem Wissenszuwachs der Lernenden im Domänenwissen gemessen. Nun ist es dem Lernenden aber nicht möglich alles auf einmal zu lernen und für ein ITS ist eine gute Sequenzialisierung eines großen Wissensblocks sehr schwierig, weil es diesen Block selbständig neu aufteilen müsste. Deshalb muss das Domänenwissen in kleinste Einheiten aufgeteilt werden, sie werden hier Wissenselemente genannt. Das ist vorteilhaft für die Anpassungsfähigkeit und Lehre der ITS. Denn ITS mit kleineren Wissenselementen zeigen in fünf von sechs fällen bessere Lehrergebnisse [2014 CHI, JORDAN and VAN LEHN]. Es wurde in dieser Arbeit untersucht welche Wissensklassen von Domänenwissen in den ITS existieren. Folgende wurden gefunden.

- Technischer Begriff: Zum Beispiel die Definition von einem für das Domänenwissen wichtigen Begriff
- Strukturmodell: Zum Beispiel die Erklärung eines Strukturaufbaus
- Auswertungsregel: Zum Beispiel ein Algorithmus, der für eine Eingabe eine Ausgabe produziert oder eine Beziehung von zwei verschiedenen Variablen
- Prozess: Beschreibung eines Ablaufs
- Beispiel: Ein konkretes Beispiel aus der Domäne zur Veranschaulichung eines Sachverhalts
- Sammlung: Eine Anhäufung von bestimmten Dingen die etwas gemeinsam haben
- Detailbeschreibung: Eine detaillierte Ausführung eines schon vorhandenes Domänenwissens

Diese Klassen von Wissenselementen können unterschiedlich in den ITSs repräsentiert werden. Verschiedene Repräsentationen ergeben die Möglichkeit das gleiche Wissen auf unterschiedliche Weise aufzuzeigen. Folgende Repräsentationen wurden für Wissenseinheiten gefunden.

- Text
- Grafik (zum Beispiel ein Diagram oder Foto)
- Audioschnitt
- Videoausschnitt
- Interaktives Lehrelement
- Domänenspezifische Sprache (zum Beispiel SQL)
- Eine speziell sortierte Ordnung, wie eine Nummerierung, eine Liste oder oder Tabelle mit möglichen Sortiervorgaben

Die Repräsentation differenziert sich nicht nur im Kommunikationsmedium, sondern auch in der Art und Weise wie etwas aufgezeigt wird. So sind Text oder Grafiken beide visuell. Eine Wissensklasse kann auf mehrere Arten und Weise repräsentiert werden werden. Ein Modell kann zum Beispiel als Grafik visualisiert, als Text beschrieben oder an-

derweitig dargestellt werden. Jede Repräsentation hat ihre eigenen Vor- und Nachteile. Die Repräsentation ist auch der ausschlaggebende Faktor für Lernende, welche Lehrressource sie nehmen, denn Lernende haben oft Präferenzen für eine bestimmte Repräsentation.

Die Größe der Wissenselemente sollte aus verschiedenen Gründen nicht nur klein, sondern minimal gehalten werden. Vorteile davon sind die Wiederverwendbarkeit, die genauere Anpassung und damit auch Sequenzialisierung von Wissenselementen und die bessere Zuordnung der Wissenselemente in deren Repräsentation und Klasse, was wiederum eine leichtere Generierung von neuem Lehrmaterial erlaubt. Bei Video- und Audioaufnahmen kommt häufiger vor, dass diese eine ganze Lehreinheit abbilden und damit nicht in kleine Wissenseinheiten aufgeteilt wurden. Dies hat seinen Einsatzzweck, dennoch ist es sehr schwierig Wiederverwendbar, weil das Video erst geschnitten werden muss. Die Idee ist es, so viele zusätzliche Daten (Metadaten) zu generieren von dem vorhandenen Daten. Um die Nützlichkeit des Lehrmaterials zu steigern ist es also das Ziel, möglichst viele Metadaten von den Wissenselementen zu erzeugen. Jede Repräsentation hat andere Anforderungen an die Generierung der Metadaten. Eine Bildanalyse hat offensichtlich andere Anforderungen als eine Textanalyse und so weiter. Einfache Beispiele für Metadaten von einem Text, die automatisch generiert werden können, ist die Anzahl der Wörter, Anzahl der Buchstaben, bekannte technische Begriffe welche auch in Textform vorliegen, Länge der Sätze mit den jeweiligen technischen Begriffen und so weiter. Nur implizit berechenbare Metadaten für Maschinen lassen sich automatisch aus den Wissenselementen in einer jeweiligen Wissensrepräsentation auswerten. Diese haben aber in bestimmten Bereichen Probleme mit der Zuordnung, um was für ein Lehrmaterial es sich handelt. Es gibt Forschungsarbeiten welche sich mit der automatischen Textanalyse beschäftigen um weitere semantische Aufschlüsse und damit Metadaten automatisch zu generieren. Zum Beispiel Textanalysen mit dem „Latent Dirichlet Allocation”-Algorithmus [2014 NICOLAY][2017 NICOLAY, GRAF VON MALOTKY and AUGE]. Trotzdem fehlen für Menschen offensichtliche Metadaten, die sich für Maschinen nur sehr schwierig oder gar nicht generieren lassen. Mehr Metadaten lassen sich von fast jedem Menschen ohne Ahnung von dem Inhalt schnell erstellen. Es sind für Menschen implizit sofort erkennbare Metadaten. Um solche Schwächen in der Generierung der Metadaten zu umgehen, könnte man Menschen einsetzen, diese einmalig vorhandenen Wissenselementen hinzuzufügen. Die Menschen müssten keine Domänenexperten sein, weil die Aufgaben simpel sind. Beispiele dafür sind die Zuteilung von vorgefertigten Kennzeichen oder die Eingruppierung in vorhandene Kategorien. Nehmen wir konkret ein Textausschnitt aus einem Lehrbuch das als Wissenselement verwendet wird: Schon anhand des Layouts, der Schriftart (auch im Vergleich mit den restlichen vorhandenen Schriftarten), der Positionierung des Textausschnitts im Kapitel und innerhalb des Absatzes, der Stelle wo es sich im Inhaltsverzeichnis befindet und vielen weiteren Indikatoren, fällt es einem Menschen nicht schwer, es auch ohne die Bezeichnung Lemma zu einer Kategorie Grundlagendefinition zuzuteilen. Auch die Bedeutung von für sich stehenden Wissenselementen wie Visualisierungen lassen sich von Menschen leichter in vorgefertigte Kategorien einteilen. Es gibt Ansätze, welche Metadaten in einem ITS notwendig sind, wie zum Beispiel LOM [2002 HODGINS and DUVAL] und SCROM [2002 ADL]. Diese Spezifikationen fokussieren sich auf den Lehrinhalt als eine Gesamteinheit und vernachlässigen damit die Adaption durch kleinere Wissenselemente. Deshalb können sie hier in einem Ansatz zur Hilfe des ITS das Domänenwissen im einzelnen zu kategorisieren nicht verwendet werden.

Beispiele für den Einsatz von Menschen als Hilfsmittel zur Generierung von Metadaten werden folgend genannt. Ein Beispiel ist die Zuordnung einer Wissensrepräsentation zu deren Art der Repräsentation, eine Audioaufnahme von einer Videoaufnahme mag simpel für den Computer sein aber nicht ob es eine Formel oder eine Grafik ist. Ein Schritt weiter wäre die Zuordnung zu Wissensklassen. Ein anderes Beispiel sind die Abhängigkeiten von

Wissenselementen zueinander. Bei Textdefinitionen von technischen Begriffen, welche in anderen Beschreibungen in Textform verwendet werden, mag es für einen Computer simpel erscheinen. Doch gibt es Abkürzungen und unterschiedliche Schreibweisen, welche für einen Menschen schnell als das gleiche erkannt werden und bei Computern zwei verschiedene Dinge darstellt. Noch viel schwieriger für Computer zu erkennen ist die Verwendung von technischen Begriffen die in Textform vorliegen und in einem Audio- oder Videoausschnitt verwendet werden. Hier stellen unterschiedliche Dialekte, Betonungen und Sprechweisen ein großes Hindernis für den Computer dar. Ein Mensch hätte es ohne den Inhalt zu verstehen einfacher aus einer Liste von Wörtern die benutzten herauszusuchen. Das letzte Beispiel benötigt schon etwas Verständnis des Domänenwissens: Wissenselemente sollen zu einem oder mehreren Themen zugeordnet werden. Solche Ideen lassen sich beliebig schwierig gestalten. Das Ziel wäre es, in den Metadaten eine Art Etikettierung für die Wissenseinheiten zu haben, sodass eine Suche des ITS gleich nach der Anfrage gefiltert werden kann. Darunter können die verschiedensten Etiketten sein, von Schwierigkeitsgrad, Komplexität, Themenbereiche oder für welche Lernenden mit bekannten Metriken es besonders geeignet sein kann.

All diese an konkret vorhandenem Wissensrepräsentationen gebundenen Metadaten sind zwangsläufig erforderlich, um konkretes Lehrmaterial zu erzeugen, um letztendlich informatives und interaktives Lehrmaterial passend zum Lernenden für eine Lehreinheit zu erstellen. Trotzdem fehlt Pädagogikwissen, welches unabhängig von dem zugrundeliegenden Lehrmaterial an Wissensrepräsentationen angewendet werden kann um daraus angepasstes Lehrmaterial zu erstellen. Diese Lücke an Wissen wird in Abbildung R-II3k-1 veranschaulicht.

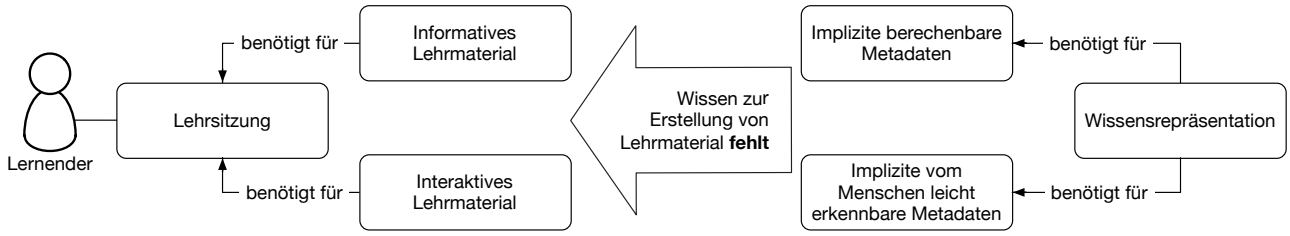


Abbildung R-II3k-1: Lücke in der automatischen Generierung von Lehrmaterial

Das Modell eines Lehrers zeigt auf, dass der Lehrer eine besondere Fähigkeit darin hat, mehr aus dem Lehrmaterial zu generieren, als er vom reinen Domänenwissen bekommt. In ihm steckt allgemeines Pädagogikwissen, welches im Gegensatz zum konkreten Pädagogikwissen von einem ITS an jeglichem Domänenwissen angewendet werden kann. Das bekannte konkrete Pädagogikwissen ist das was wir von ITS kennen, es sind zusätzliche Beschreibungen passend zu dem vorhandenen Domänenwissen: Das vorhandene Domänenwissen passend für den aktuellen Lernenden in seinem jetzigen Wissensstand und Zustand zu filtern und Interaktionsvorschläge zu kennen bei bestimmten Verhaltensweisen eines Lernenden. Es soll helfen das Domänenwissen zu vermitteln, indem das richtige Domänenwissen zum richtigen Zeitpunkt auf die passende Weise erklärt wird. Konkretes Pädagogikwissen kann also als eine Annotation von Domänenwissen angesehen werden, anstatt nur Fakten basierend zu lehren. Es gibt eine Art abstrahiertes und allgemeines Pädagogikwissen welches dagegen unabhängig vom Dömanenwissen angewendet werden kann. Ein ITS arbeitet bisher nur mit konkretem schon von den jeweiligen Lehrern erzeugtem Pädagogikwissen, welches zum Beispiel dazu verwendet wird Lehreinheiten durch Sequenzialisierung des Domänenwissen zu nutzen. Die Lösung sind damit Lehranreicherungsregeln. Es sind Regeln, welche abstrahiertes Pädagogikwissen darstellen und nach denen vorgegangen werden kann, um das benötigte an Domänenwissen gebundene konkrete Pädagogikwissen zu erzeugen, aus dem sich dann angepasste Lehreinheiten aus interaktiven und informativem Teil erstellen lassen können. Solche domä-

nenübergreifenden Regeln werden in der Didaktik definiert. Diese müssten für ein Computersystem definiert werden, um auch ihm eine ähnliche Fähigkeit zu definieren. Der Einfachheit halber wird das konkrete Pädagogikwissen nur Pädagogikwissen genannt. Die Lehranreicherungsregeln müssen ebenfalls manuell erstellt werden wie das Pädagogikwissen und zwar von keinem Lehrer des Lehrstoffs sondern Didaktikern. Der Vorteil ist die große Wiederverwendbarkeit. Anstatt immer wieder neues Pädagogikwissen zu erzeugen werden die Lehranreicherungsregeln häufiger verwendet. Selbst wenn das Ergebnis des automatisch erzeugen Pädagogikwissen nicht die Qualität von händisch erstelltem Lehrmaterial hat, so wird dies ab einer Minimalqualität von der Kosteneffizienz und Geschwindigkeit überwogen. Andere Gründe für den Einsatz könnten sein, dass es womöglich einfacher und wirtschaftlicher ist es breit gefächerte Lehranreicherungsregeln zu entwerfen, weil damit auch Lehrmaterial für sehr kleine Domänen abgedeckt sind, für das es sich nicht lohnen würde nur dafür manuell Pädagogikwissen zu erstellen. Didaktiker, die keine Ahnung von dem jeweiligen Domänenwissen haben könnten trotzdem sehr gute Lehranreicherungsregeln schreiben und Domänenexperten ohne Lehrerfahrung könnten sehr detaillierte Metadaten verfassen. Die Kombination daraus könnte sehr hohe Qualität aufweisen.

Lehranreicherungsregeln werten klar definierte Bedingungen von Wissenselementen und derem Metadaten aus. Ihr Ergebnis ist Pädagogikwissen. Sie reichern also vorhandenes Faktenwissen mit Annotationen an, welche nur für die Lehrseite relevant ist. Sobald das Pädagogikwissen erzeugt ist sind Metadaten nur noch unwichtige Zusatzinformationen. Man sollte sie dennoch behalten, falls man die Lehranreicherungsregeln verbessert um auch wieder verbessertes Pädagogikwissen zu erzeugen. Der Lernende will direkt nur Domänenwissen lernen und wird nur indirekt durch ein verbessertes ITS-Verhalten durch bessere Metadaten, Lehranreicherungsregeln und damit auch Pädagogikwissen beeinflusst. Beispiele für allgemeine Lehranreicherungsregeln sind, dass ähnlicher Lehrstoff zusammen gelernt werden sollte, der Schwierigkeitsgrad aufsteigen sollte, zuerst von Vorwissen unabhängiges Wissen gelernt werden sollte und so weiter. So kann dies mit der Bedingung nur für eine Domäne zu gelten auch sehr speziell werden. Auch hier müssen die Regeln nicht zu hundert Prozent stimmen, das Pädagogikwissen verändert das Verhalten von dem ITS, deren Qualität führt aber nicht dazu, dass der Lernende falsche Fakten lernt, sondern nur ineffizient. Es kann sein, dass Pädagogikwissen nur bei bestimmten Gruppen von Lernenden besonders gut funktioniert. Perfekt wäre wenn das Pädagogikwissen das Verhalten des ITS an jeden Lernenden anpassen könnte.

Damit kann Pädagogikwissen dynamisch an einem Profil von einem Lernenden erzeugt werden. Es ist ein erster Schritt Lehrmaterial von einem ITS in einer allgemeinen und unabhängigen Art und Weise zu erzeugen. Das Profil ist auch eine Generierung von Wissen aus dem vorhandenem Lernendenmodell. Es macht Annahmen über das Wesen und das Wissen des Lernenden im Gesamten, dabei nähert sich das Profil mit einem größer werdendem Lernendenwissen an der Realität an. Um es dem Lernenden nicht unnötig mühselig zu machen, kann dieser natürlich seinen aktuellen Wissenszustand auch eigenmächtig erhöhen und damit das Profil künstlich erhöhen. Wenn an Übungen vom ITS gemerkt wird, dass diese nicht zustimmen, wird sich die Heuristik automatisch anpassen. Letztendlich gibt es mehrere Schichten von Wissen die teilweise generiert werden um eine Lehreinheit zu generieren, siehe dazu Abbildung R-II3k-2. Das implizierte Wissen muss zuerst, unter Umständen mit Hilfe von Menschen für die Metadaten, generiert werden. Dann kann konkret an das gewählte Domänenwissen Pädagogikwissen generiert werden um letztendlich informatives und interaktives Lehrmaterial aus dem erstellen Pädagogikwissen und dem Lernendenprofil zu erstellen und dann in einer Lehrsitzung einzusetzen.

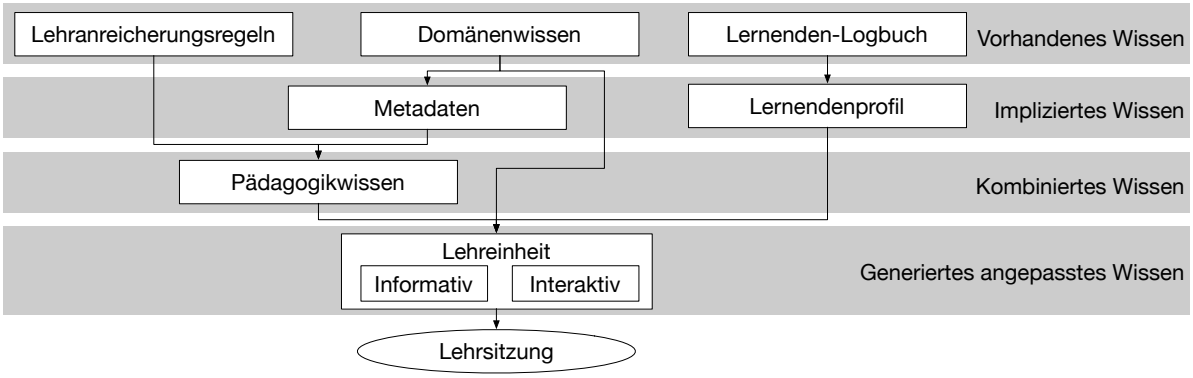


Abbildung R-II3k-2: Wissensschichten für die Lehrinhaltgenerierung

Das Domänenwissen wird verwendet um die Metadaten von Mensch und Maschine zu implizieren. Anhand der Metadaten, der Lehranreicherungsregeln und des Domänenwissen kann das Pädagogische Wissen generiert werden. Aus dem Lernendenmodell, wie zum Beispiel einen Interaktionslogbuch, kann das Profil impliziert werden. Mit Domänenwissen, Pädagogikwissen und Profil kann Lehrinhalt generiert werden. Der Lehrinhalt wird in einer Lehreinheit kombiniert und kann in einer Lehrsitzung verwendet werden.

Beispiele für den Inhalt der verschiednen Wissensbausteine sind folgend aufgezählt.

- Lernenden-Logbuch: Der Lernende hat Addition erklärt bekommen und 90% der gestellten Aufgaben darin lösen können. Der Lernende macht häufig Aufgaben, in denen nicht nur konkrete Zahlen stehen, falsch. Der Lernende hat überdurchschnittlich schnell Biologie gelernt in Wissenselementen mit Animationen.
- Profil: Der Lernende versteht Algebra noch nicht genügend für die Addition. Er ist besser beim Lernen wenn vorher etwas in einer Animation gezeigt wurde.
- Domänenwissen: Formeln, Definitionen und so weiter als Repräsentation des Domänenwissen
- Lehranreicherungsregeln: Der Schwierigkeitsgrad innerhalb einer Lehreinheit sollte gleich bleiben. Es sollten alle Abhängigkeiten beherrscht werden, die für ein Wissenselement benötigt werden. Der Schwierigkeitsgrad ohne Abhängigkeiten ist null, jede weitere Abhängigkeit erhöht sie um eins.
- Metadaten: Eine Sammlung von Metadaten die sowohl maschinell generiert als auch manuell hinzugefügt worden ist über das Domänenwissen, Addition gehört zur Mathematik und hat den leichtesten Schwierigkeitsgrad mit konkreten Werten und einen höheren mit Buchstaben.
- Pädagogikwissen: Für den informativen Teil der Lehrsitzung sind es Zusammenhänge zwischen den Lehrmaterialien zueinander (benötigtes Vorwissen/gute Grundlage für), deren geeignete Unterrichtsweise und deren Komplexität besonders wichtig. Als Beispiel für einen Zusammenhang sei die übliche Reihenfolge von Addition und dann Multiplikation genannt. Es sollte zuerst Addition ohne Algebra gelernt werden. Die Ganzen Zahlen und Subtraktion sind gutes Folgewissen, das nach Addition gelernt werden kann. Abstraktion von Zahlen auf Platzhalter hat einen viel höheren Schwierigkeitsgrad. Personen, die die Addition nur mit konkreten Zahlen beherrschen, sollten eine ausführliche Einführung in die Algebra bekommen. Für den Informativen Teil sind es Rückmeldungen oder Fehlinterpretationen.
- Lehreinheit: Ein Pfad durch die Wissenselemente in einem fest gestecktem Thema. Hier die Addition als Thema in einem einzigen Schwierigkeitsgrad. Einmal die Erklärung und einmal (generierte) Übungen für Algebra.

II.4. Prozess in einem ITS

II.4.a. ITS-Lehrprozess

Von dem Komponentenmodell wird ein Modell des Lehrprozesses abgeleitet. Der Ablauf zum Lernen einer Lehrsitzung mit begrenztem Zeitrahmen mit einem ITS kann in mehrere Aktionen aufgeteilt werden. Dabei wird immer wieder der gleiche Ablauf ersichtlich, siehe Abbildung R-II4a-1.

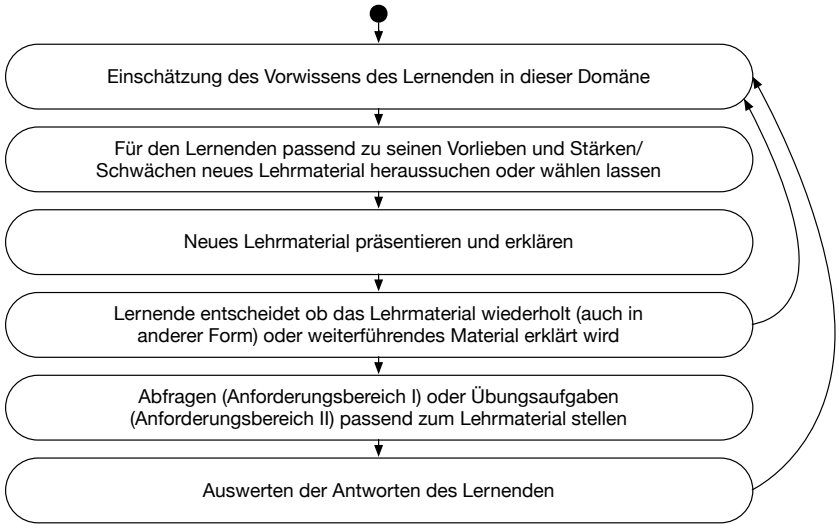


Abbildung R-II4a-1: Prozess des Lernens in Hinblick auf ITSs

Inhaltlich wurden folgende Lernphasen bei ITSs gefunden ohne dass diese explizit erwähnt wurden.

- Wissen vermitteln: Anhand von Beispielen, Definitionen, Metaphern wird das Domänenwissen dem Lernenden gelehrt
- Wissen festigen: Mit Übungen in denen das Wissen angewendet werden muss
- Wissen überprüfen: Testen ob das Wissen wie vorgesehen gelernt wurde.
- Wissen erweitern: Vorhandenes Wissen wird als vermittelt angesehen und darauf kann aufgebaut werden

Nach Roth gibt es sechs verschiedene Lernphasen, welche eingehalten werden sollten für einen effizienten Lernablauf, siehe dazu Tabelle R-II4a-2. Auch hier zeigt sich, dass erst ein Grundwissen oder Lösungsalgorithmus gelehrt wird, bevor geübt wird.

Stufe	Beschreibung
1	Stufe der Motivation
2	Stufe der Schwierigkeiten
3	Stufe der Lösungen
4	Stufe des Tuns und Ausführens
5	Stufe des Behaltens und Einübens
6	Stufe des Bereitstellens, der Übertragung und der Integration des Gelernten

Tabelle R-II4a-2: Lernphasen nach Roth [1957 ROTH]

Diese Anforderungen an den Lehrprozess muss für ITS angepasst werden, weil ein ITS nicht nur ein digitalisierter Lehrer darstellt. Es gibt Architekturen, die schon versucht haben einen Prozess für ITS sehr grob zu definieren, dieses aber in der Architektur selbst abbilden [2014 IMRAN] oder [2000 ZOUAQ, FRASSON and ROUANE]. In eine Softwarearchitektur wird per Definition kein Prozess abgebildet. Dies führt zu einigen Problemen und Missverständnissen. Für eine Software wie ein ITS sollte der zu entwickelnde formalisierte Prozess so aufgebaut sein, dass er verbesserungsfähig ist und nahtlos mit der ent-

wickelten Architektur zusammenarbeitet. Um ein ITS in einer Implementierung nicht unnötig einzuschränken und übertragbar auf andere Architekturen zu sein muss er allgemein genug sein. Es gibt verschiedene Phasen die man so grob auch von dem Prozess mit anderen ITSs übernehmen kann. Um die Adaptivität zu gewährleisten und damit das Verständnis des Lehrstoffs zu erhöhen, müssen abhängig von den Handlungen der Lernenden an bestimmten Stellen unterschiedliche Wege gegangen werden. Deshalb ist der Prozess ereignisorientiert. Wobei die meisten Ereignisse interaktive Aktionen des Lernenden darstellen. Auch ohne den Lernenden können proaktive und reaktive Ereignisse auftreten.

II.4.b. Sequenzieller Ablauf

Der Prozess ist relativ linear gehalten. Auf den ersten Blick scheint dies bei der Initialisierung der Datenbanken unnötig. Bei einem parallelen Ansatz ist die Übersetzung des Prozesses in andere formale Beschreibungen schwieriger. Es gibt drei Wissenskomponenten welche Daten bereitstellen müssen. Die Probleme bei der Darstellung werden am Beispiel eines Zustandsautomaten sichtbar. Der Umsetzungsversuch mit einem Endlichen Automaten ergab Probleme durch eine Kombination von diesen drei Wissenskomponenten, wenn diese in beliebiger Reihenfolge initialisiert werden können und erzeugte einen Automaten, welcher sehr komplex wurde. Es entstand ein sehr großes Diagramm um den Automaten darzustellen. Die Problematik wird schnell schon in der ersten Aufgabe ersichtlich, ein Ausschnitt welches die Problematik zeigt ist in Abbildung R-II4b-1 zu sehen. Die Farbe wurde zur besseren Unterscheidung künstlich hinzugefügt um die drei Komponenten besser zu unterscheiden: Domänenwissen in grün, das Lernendenwissen mit dessen Profil in gelb und das Pädagogikwissen in pink. Die einzelnen Aufgaben werden später noch detailliert aufgeführt. Die Visualisierungen soll nur aufzeigen, dass der endliche Automat keine gute Wahl für diese Beschreibung darstellt.

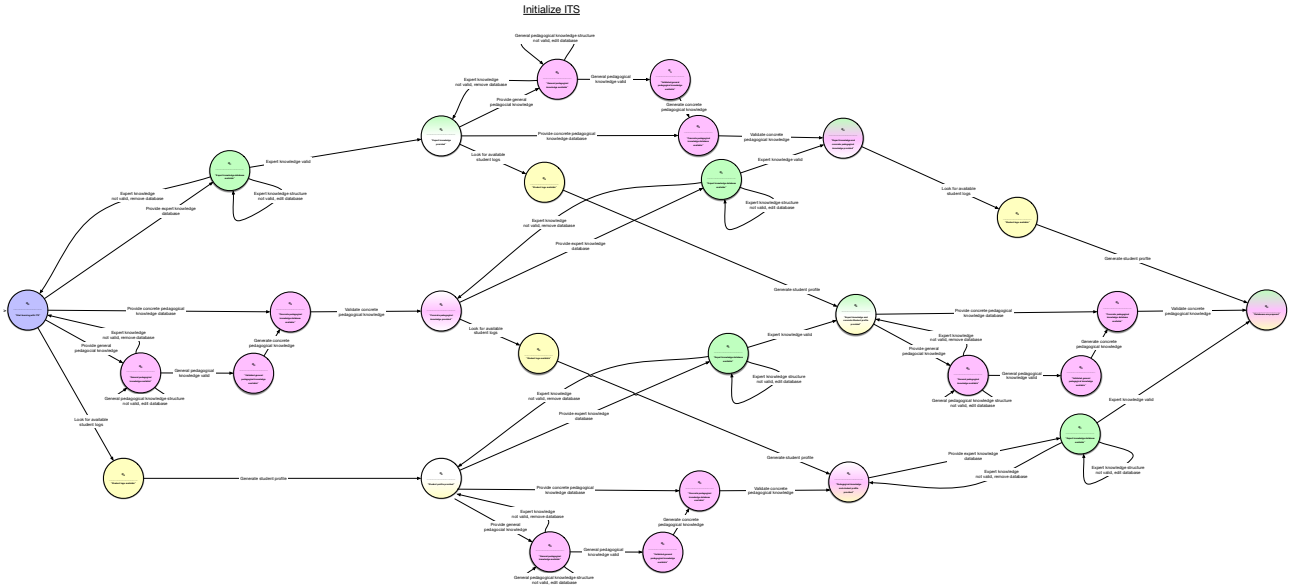


Abbildung R-II4b-1: Ausschnitt ITS-Prozess als endlicher Automat

Parallelität stellt dabei ein Modellierungsproblem für einen endlichen Automaten dar, weil die Anzahl der Zustände schnell ansteigt und damit ein einfacher Zusammenhang schwer verständlich wird. Dieses Problem wird auch Zustandsproblem genannt und führt dazu, dass der endliche Automat eine ungünstige Wahl für diese Art von Beschreibung ist. Petrinetze sind für die Prozessanalyse der automatisierten Lehre besser geeignet. Es wurde mit einem Petrinetz eine andere Ansicht für den ITS-Lehrprozess gegeben. Das gesamte Petrinetz ist in Abbildung R-II4b-2 zu sehen.

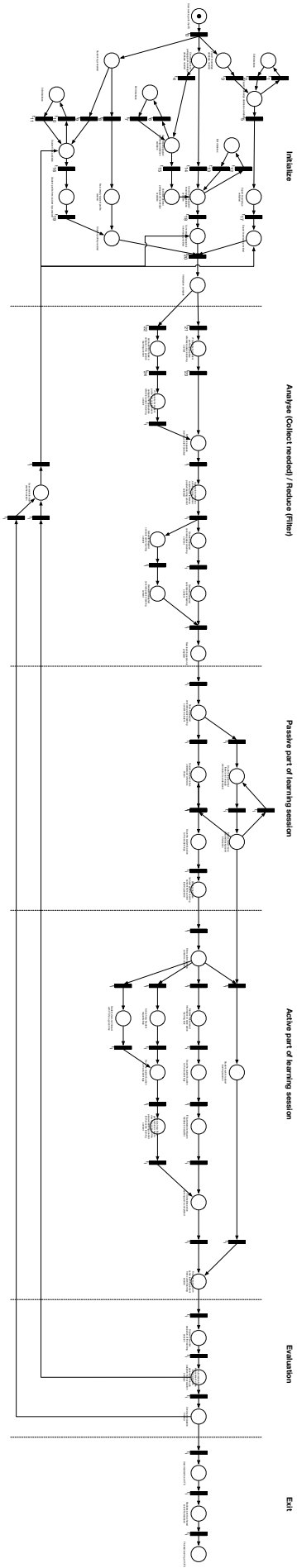


Abbildung R-II4b-2: ITS-Lehrprozess als Petrinetz

Das Petrinetz erlaubt eine formale Darstellung, die in Werkzeugen interaktiv verwendet werden kann. Die Implementierung eines Colored Petri Nets erlaubt es mit dem Werkzeug „CPN Tools 4.0“ auch eine aktive Simulation des Netzes, siehe Abbildung R-II4b-3. Dahingehend wurde der Prozess vereinfacht, indem die Reihenfolge strikter gehalten wurde. Dadurch gibt es ein einfacher verständliches Aktivitätsdiagramm in UML mit reduzierten Optionen in der Abfolge der Aktivitäten und damit auch reduzierter Komplexität.

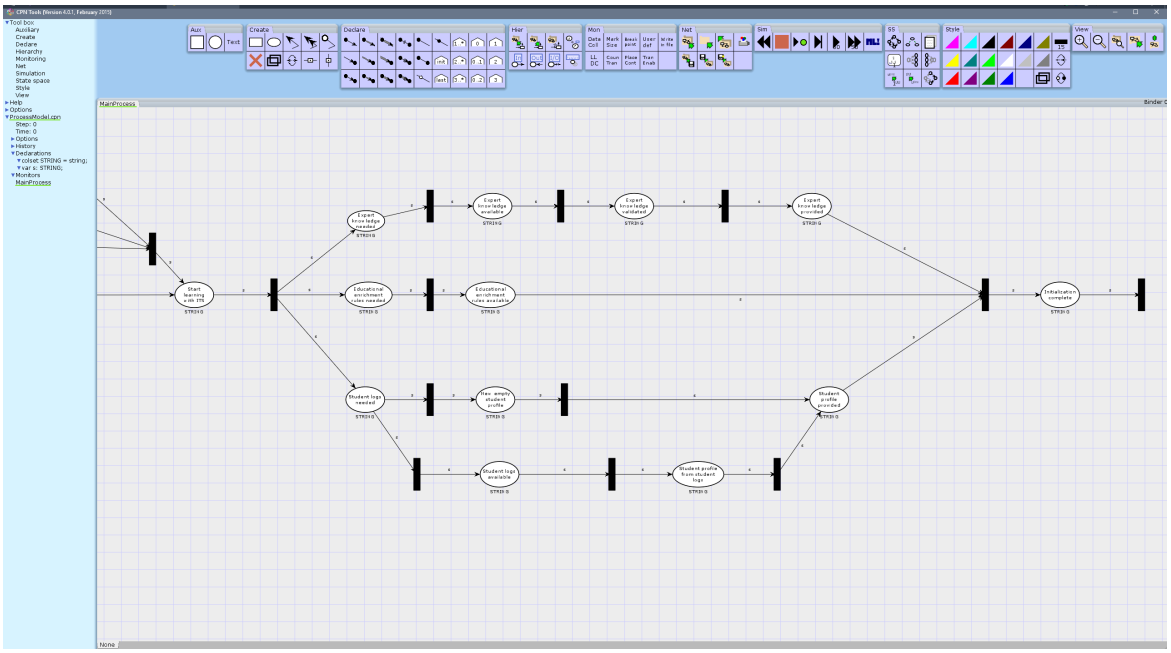


Abbildung R-II4b-3: Aufbau des Petrinetzes im CPN-Tool

II.4.c. Aktivitätsdiagramm

Es wurde sich letztendlich für ein UML Aktivitätsdiagramm für die Prozessbeschreibung entschieden. Der grobe Aufbau des Prozesses für ITS wird in Abbildung R-II4c-1 gezeigt. In ihm sind mehrere Aktionen (offizielle Bezeichnung), die jeweils noch genauer beschrieben werden. Man kann aber schon erkennen, dass es in den ersten drei Aktionen eine Initialisierung gibt und dann eine Schleife aus den hinteren vier Aktionen. Bei jedem Durchlauf wird das angepasste Lernendenwissen und die während der Benutzung statischen Datenbanken von Domänen- und Pädagogikwissen verwendet um eine Lehreinheit zu erstellen die der Lernende in einer Sitzung durcharbeitet. Auf diese Datenbanken, den persistenten Zustand, hat der Lernende keinen direkten Zugriff. In welchem Schritt sich das ITS befindet und wo sich der Lernende befindet, beziehungsweise was er dort aktuell getan hat, wird im temporären Zustand gespeichert. Der temporäre Zustand kann von dem Lernenden in gewissen Grenzen bestimmt werden und ermöglichen ihm die Navigation und die Auswahl der Interaktionen. Zu guter Letzt kann der Lernende noch eine subjektive Evaluation abgeben. Bestimmte Aktionen können auch die Möglichkeit besitzen nicht zur nächsten Aktion zu leiten, sondern zu einer vorherigen. Diese Abbrechen-Option wird nicht überall benutzt, weil es den Lernenden verleiten könnte schneller aufzugeben.

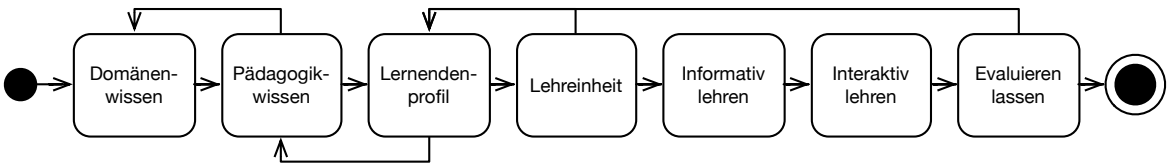


Abbildung R-II4c-1: Grobe Einteilung des ITS-Prozesses [2017b GRAF VON MALOTKY, NICOLAY and MARTENS]

Statische Daten initialisieren

Die erste Aktion sorgt für das korrekte initialisieren des Domänenwissens. Es befindet sich, je nach ITS auf der Festplatte oder muss vom Server geladen werden. Mit seinem Zugriff muss auch der Inhalt korrekt geladen werden. Die Überprüfung kann sehr unterschiedlich ausfallen, solange anschließend sichergestellt ist, dass Domänenwissen verfügbar und bereit für Anfragen ist, siehe Abbildung R-II4c-2. Auf die gleiche Art und Weise muss das Pädagogikwissen bereitgestellt werden. siehe hierzu Abbildung R-II4c-3.

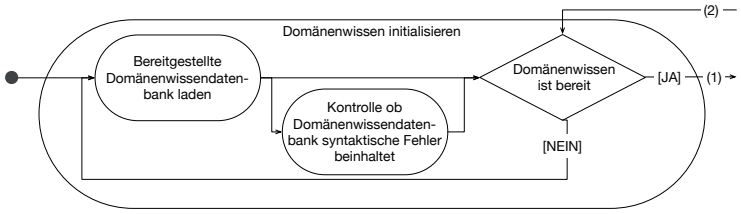


Abbildung R-II4c-2: Domänenwissen initialisieren

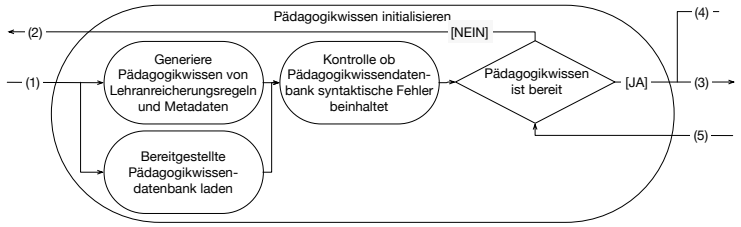


Abbildung R-II4c-3: Pädagogikwissen initialisieren

Dynamische Daten initialisieren

Da nun die statischen Datenbanken verfügbar sind, welche sich während des Lehrprozesses nicht mehr ändern, werden als nächstes die an den Lernenden angepassten Daten generiert. Das Lernendenwissen als eine Art Logbuch über die Aktionen die dieser getätigt hat muss ausgewertet und zusammengefasst werden, siehe Abbildung R-II4c-4. So entsteht das Profil, als eine Interpretation die simpler verwendet werden kann um Lehrmaterial anzupassen als das reine Lernendenwissen.

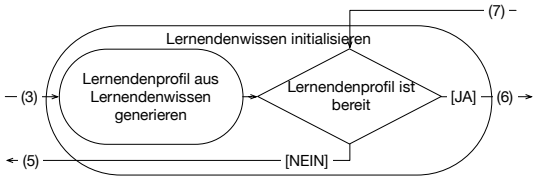


Abbildung R-II4c-4: Lernendenwissen initialisieren

Genau das und mehr wird in der nächsten Aktion gemacht, siehe dazu Abbildung R-II4c-5. Es wird zunächst nach informativen Lehrmaterial für das Profil gesucht, dann muss davon eine Menge von Material gefunden werden, die für eine Lehrsitzung zeitlich und inhaltlich zusammen passt, dies wird durch das Pädagogikwissen gemacht. Entweder komplett automatisch oder man überlässt die Entscheidung des Themas dem Lernenden. Erst dann kann passend dazu auch interaktives Lehrmaterial gewählt werden, welches zum Beispiel Übungsschablonen oder Fragestellungen mit Lösungswegen sein können, um zu überprüfen, ob das informative Lehrmaterial verstanden wurde. Mit der weiteren Verwendung des aktuellen Profils und der Historie der Profile als Entwicklungshistorie kann auch das informative und interaktive Lehrmaterial konkretisiert und weiter an den Lernenden angepasst werden. Ein Beispiel bei dem das ITS sich mittels des Profils anpasst wäre: Die Menge an Informationen und welche Wissensrepräsentation im informativen Lehrmaterial gewählt wird hängen davon ab, was der Lernende schon mittels des ITS gelernt und evaluiert hat. Es wird immer weniger Basiswissen beinhalten und nur

noch Wissen in der Art und Weise die dem Lernenden zusagt. Einfache Berechnungen für das Profil ließen sich zur Laufzeit berechnen, doch ein zur Laufzeit komplett dynamisches Profil hätte Performanzeinbußen. Der Vorteil eines Profils zur Laufzeit neu zu berechnen wird als gering eingeschätzt. Das Profil wird als einzige Quelle für die Anpassung (Schwierigkeitsgrad, die Komplexität, die Art des Lehrmaterials, das Verhalten und so weiter) genutzt und vereinfacht damit die Komplexität der Abhängigkeiten selbst. Beispielsweise können Erklärungen verkürzt und damit weniger Detailinformationen oder Hinweise enthalten, wenn der Lernende besonders gut in dem Bereich ist.

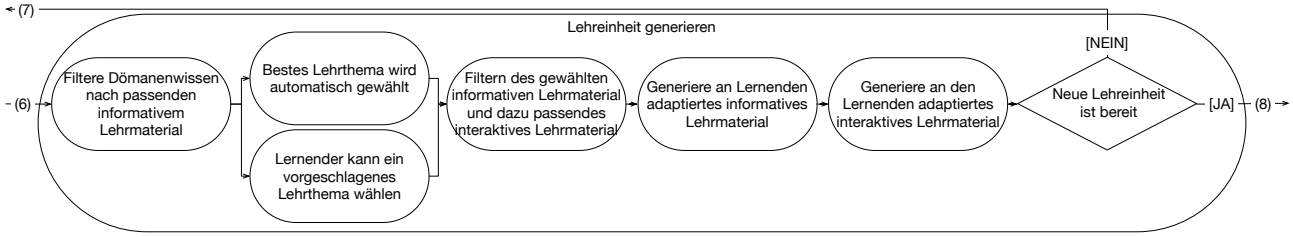


Abbildung R-II4c-5: Lehreinheit generieren

Lehrsitzung

Bei der Lehrsitzung unterscheidet man informative und den interaktive Anteile. Der Unterricht ist für die beiden Lehrmaterialienarten verschieden. Die Lehre des informativen Lehrmaterials ist zeitlich zuerst, weil sie das Domänenwissen unterrichtet, welches notwendig ist, um die Übungen nicht ahnungslos zu beginnen, siehe Abbildung R-II4c-6. Es wird immer Stück für Stück das in der vorherigen Aktion gewählte informative Lehrmaterial präsentiert. Nachträglich wird abgefragt ob der Lernende glaubt, es verstanden zu haben oder ob mehr grundlegenderes informatives Lehrmaterial präsentiert werden soll. Es ist wichtig, dass nur für die Lehrsitzung relevantes Wissen nachgeschlagen werden kann um fokussiert zu lernen. Es wird in einem Bereich gelernt der auf Domänenwissen aufbaut die dem Lernenden laut Lernendenwissen bekannt sind, aber immer nachgefragt werden können. In allen Fällen wird das Lernendenwissen auch als ein Modell des Wissensstands aktualisiert. Wurde alles präsentiert ist diese Aktion beendet.

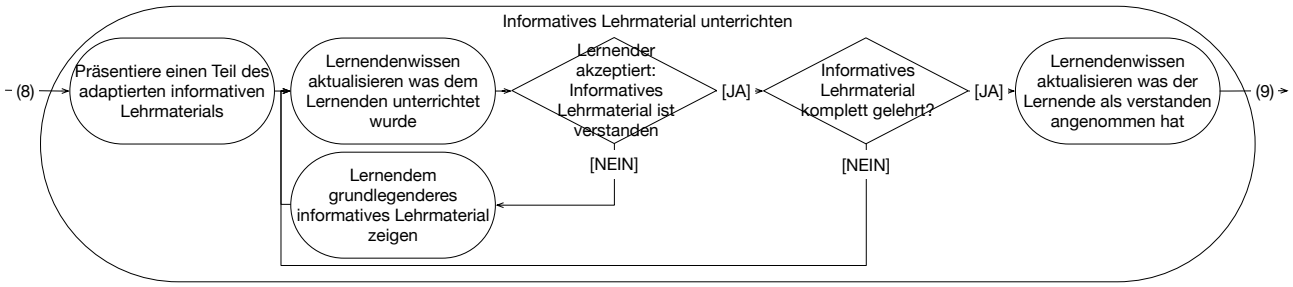


Abbildung R-II4c-6: Informatives Lehrmaterial unterrichten

Die Lehre des interaktiven Lehrmaterials verlangt von dem ITS mehr ab. Der Lernende hat jetzt mehr Möglichkeiten mit dem ITS zu interagieren, siehe Abbildung R-II4c-7. Der Lernende muss Aufgaben lösen um das Verständnis des Domänenwissens zu erhöhen und um dem ITS zu überprüfen ob der Lernende es auch nicht falsch interpretiert hat. Es gibt drei grobe Aktionsbereiche die vom Lernenden ausgeführt werden können: Lösungsschritte abgeben, Hilfe nachfragen und nichts tun. Sowohl bei keiner Aktion als auch bei Nachfragen einer Hilfe wird von dem ITS Hilfe angeboten und abgespeichert, dass diese benötigt wurde. Diese Unterscheidung nennt man auch reaktive oder produktive Hilfe. Laut Razzaq [2010 RAZZAQ and HEFFERNAN] ist es besser die Hinweise auf Nachfrage anzuzeigen als sie proaktiv zu zeigen, besonders bei Lehrmaterial bei denen viele unterschiedliche Hinweise passen würden. Bei einer Interaktion von einem Lernenden als ein Lösungsschrittversuch, wird dieser Lösungsschritt ausgewertet und bewertet und darauf-

hin eine Antwort vom ITS gegeben, bis der interaktive Teil durch ein Abbrechen oder lösen des Problems (durch den Lernenden selbst oder dem ITS falls der Lernende es nicht schafft) stattgefunden hat.

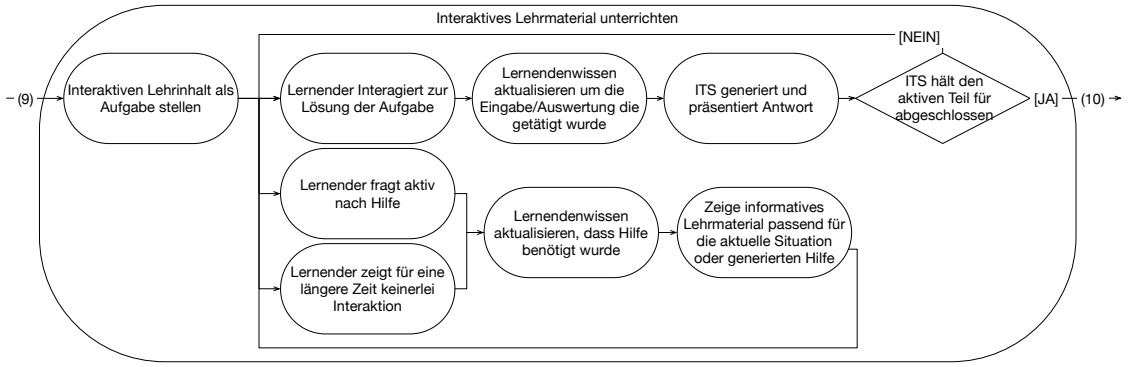


Abbildung R-II4c-7: Interaktives Lehrmaterial unterrichten

Evaluierung

Im letzten Abschnitt des Prozesses hat der Lernende die Möglichkeit seine Bewertung abzugeben über das Lehrmaterial oder die Interaktion mit dem ITS, siehe hierzu Abbildung R-II4a-8. Die Evaluierung gibt einen Aufschluss über die bewusste subjektive Bewertung des Lernenden. Die psychosozialen Faktoren können einen großen Einfluss auf die Lernfähigkeiten haben und sollten daher gesammelt werden. Für spätere Auswertung und Hinweise für die Autoren, einer womöglichen zusätzlichen automatischen Anpassung der Lehrmaterials, als subjektives Vergleichskriterium zwischen den ITSs oder als Sammlung für die Forschung in der Didaktik. Die Schleife für eine neue Lehreinheit kann begonnen werden oder das ITS beendet werden.

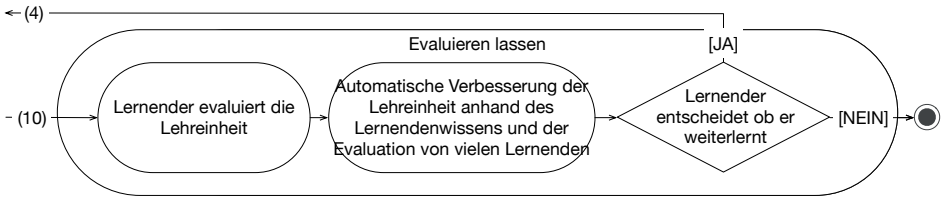


Abbildung R-II4c-8: Evaluation des Lernenden bekommen

II.5. Wahl der Konzeptgrundlage

II.5.a. Konzeptgrundlage

Um herauszufinden was ein gutes ITS ist wurden die theoretischen Grundlagen von ITS erarbeitet. Darauf aufbauend konnten die Lernmodelle und die Lerntheorien recherchiert werden von dem die beste Wahl getroffen werden muss. Dann wurde analysiert welche Lehrsysteme existieren und wie sich ITSs mit ihrer Geschichte darin eingliedern. Es ist damit schon definiert was ein ITS ausmacht. Doch welche Techniken sich bewährt haben um ein solches effektiv und effizient zu entwickeln fehlt. Diese werden hier folgend besprochen, um darauf aufbauend eine gute ITS-Softwarearchitektur entstehen kann. Das heißt nicht, dass jedes ITS diese Wahl treffen muss um noch als ITS zu gelten oder effizient zu sein, sondern beschreibt lediglich eine effiziente Form eines allgemeinen ITS-Konzepts.

Gute Wahl einer Lerntheorie

Die Lerntheorien zeigen die grundverschiedenen Ansichten wie gelernt wird. Dies hat starke Auswirkungen auf die Lehrmethode. Fast alle ITS folgen dem Kognitivismus. Das war gerade eine große Änderung von alten Lehrsystemen zu ICAI und dann ITS, der Wechsel vom Behaviorismus zum Kognitivismus. Ein reines Wiederholen auf Basis des Behaviorismus war nicht ausreichend. In das andere Extrem, dem Konstruktivismus, entwickeln sich die ITSs allerdings auch nicht, weil es zu frei gestaltet ist. Man erkennt einen Gradienten in dessen Mitte sich der Kognitivismus befindet und an den Enden Konstruktivismus und Behaviorismus. Der Grad der Freiheit und des selbstverantwortlichen Lernens wird darin abgebildet, weniger bedeutet mehr in Richtung Behaviorismus, mehr in Richtung Konstruktivismus. Softwareseitig ist es einfacher Lehrprogramme des Behaviorismus zu implementieren, weil Lernpfade vordefiniert sind und generell der Lernende viel Eingeschränkter in seinen Handlungen ist. Alle Möglichkeiten des Lernprozesses sind vorgegeben und die Evaluation einer Antwort lässt sich auf richtig und falsch oder gering abzählbaren Antworten runterbrechen. Es muss weniger darauf eingegangen werden, wie sich der Lernende verhält sondern kann statisch von mehr Vorbedingungen ausgehen. Der Konstruktivismus erfordert zwar weniger Hilfestellungen von einer Software, aber soziales Verständnis. Da dies Programmen noch immer überaus schwer fällt es von der Machbarkeit schon raus. ITS implementieren gerne „Scaffolding“, eine Lehrmethode bei dem die Hilfestellungen abnehmend gestaltet ist. Anfangs gibt es mehr Hilfestellung und Assistenz, die immer weiter reduziert wird umso besser der Lernende wird. Ganz ohne Hilfestellung und Erklärung, wie bei einem extremen Lehrprogramm im Konstruktivismus, wird es nicht gehen, da es schwierig ist, zeitnahe Lernfortschritte in einer speziellen Wissensfeld zu erlangen und dabei auch die Motivation aufrecht zu erhalten. Denn einer Software wird es, selbst bei einem zu einem Menschen gleichwertigem sozialen Verständnis und Interaktionsvermögen, schwieriger fallen ein so starkes externes Feedback zu geben wie ein Mensch, auch weil die Autorität eines Computers und der Respekt zu diesem nicht so hoch ist wie die zu einem menschlichen Lehrer. Zudem kann ein Lernender schnell frustriert werden, wenn er ohne anfängliche Hilfestellung ein Problem vorgesetzt bekommt und es erst selbst erforschen muss. Das Ergebnis ist der Kognitivismus als Lerntheorie der Wahl für ein ITS. Passend zu der Mehrheit der implementierten ITSs.

Gute Wahl eines Lernmodells

Es gibt unterschiedliche Lernmodelle welche auf den Lerntheorien aufbauen. Da, wie schon im vorherigen Punkt beschrieben, der Kognitivismus am passensten ist um ein ITS umzusetzen, fällt die Wahl des Lernmodells einfacher. Die zwei bekanntesten Lernmodelle die dem Kognitivismus folgen sind „Lernen am Modell“ und „Lernen durch Einsicht“. Für ein ITS muss das Lernendenmodell so akkurat wie möglich sein, weil es sich nur an diese Informationen adaptiert. Es muss also überprüfbar sein ob das Lernendenmodell korrekt ist. Wenn man nur die Ergebnisse auf ihre Korrektheit überprüft, dann weiß man nur ob die Testmenge richtig beantwortet wurde, allerdings nicht, ob der Lernende es bei anderen Fragen über den gleichen Inhalt auch korrekt beantworten würde. Beim „Lernen durch Einsicht“ kann das Problem umgangen werden, indem man laut Lernendenmodell angeblich bereits verstandenes Wissen durch Abfrage des kognitiven Modells überprüft. Zusätzlich kann die Fertigkeit der Umsetzbarkeit dieser Einsicht durch Übungen auf die korrekte Antwort überprüft werden. Dies darf aber nicht der einzige Test bleiben. Es ist so als würde ein Lehrer erwarten, dass der Lernende erklärt wie er es verstanden hat, anstatt Aufgaben zu lösen. Beim „Lernen am Modell“ ist diese Überprüfung schwieriger, da durch Nachahmung gelernt wird. Die Imitation des gleichen Verhaltens führt aber nicht zwangsläufig zum gleichen kognitivem Modell, wie es vom ITS erwartet werden würde. Es gibt in diesem Lernmodell sogar zwei Kategorien wie beim Lernenden das gespeichert sein kann: Bildhafte Repräsentation und symbolhafte/sprachliche Repräsentation. Das „Lernen durch Einsicht“ ist daher besser geeignet für die Entwicklung eines ITS, weil es nicht von vorn herein davon ausgeht, dass der Lernende ein anderes kognitives Modell haben kann, als jenes was von dem ITS propagiert wurde. Es ist ein vorhersehbareres und leichter kontrollierbareres Modell. Falls sinnvoll kann in einem ITS mit dem „Lernen durch Einsicht“ als Lernmodell mehrere Varianten des Wissens hinterlegt werden, zwischen denen abhängig von den Vorlieben und Anforderungen eine ausgesucht wird.

Gute Wahl einer Lehrsystemklassifikation

Die verschiedenen Klassen der Lehrsysteme wurden in einem vorherigen Kapitel aufgezeigt, siehe Abbildung R-II2a-2. Der Grad der Interaktionsfreiheit ist das Merkmal, welches zwischen den Klassen stark variiert und ist deshalb ein gutes Kriterium um daran auszumachen zu welcher Lerntheorie und Lernmodell ein System zugehörig ist. Die Lerntheorien bilden dabei drei Punkte auf einer linearen Entwicklung von Lernmodel. Es gibt zwei große Richtungen in die ein Lernmodell sich neigen kann. Ein strikteres Modell, welches den Lernenden mit klaren Regeln und Strukturen vorgibt was genau wie zu lernen ist. Dazu existiert dann ein Gegenpol, welches dem Lernendem möglichst viele Freiheiten gibt. Zu guter letzt auch ein adaptives System, welches versucht die Vorteile aus beiden Welten zu implementieren.

Nun wird eine Einteilung der Interaktivität welcher ein Lernender zu einem Lehrsystem hat vorgenommen, siehe dazu Tabelle R-II5-1, bei dem eine höhere Interaktion mit einem höheren Grad gegeben ist. Laut Biggs ist es die Interaktion mit dem System, welches das Lernen verursacht. Durch die Stufen von Biggs wird ein Lehrsystem bevorzugt, welches mehr Interaktion von dem Lernenden erfordert. Damit ist Klasse „Drill and Practice“ und „Hilfe“ nicht das angestrebte Ziel für ein ITS. Die Anforderungen von der dritten Stufe des Biggs Modells erfordern auch eine Rückmeldung über das geübte Wissen zu geben. Damit fällt die Wahl auf eine der „Tutor"-Klassen. Einen hohen Grad an Interaktion mit dem Lehrsystem selbst und dessen Hilfe etwas zu Lernen bekommt man durch ein System der Klasse „Aktiver Tutor“. Dies ist nicht der Fall bei einem System der Klasse „Passiver Tutor“. Deshalb ist die Lehrsystemklasse „Aktiver Tutor“ die bessere Wahl für ein ITS, weil dort nicht die Ressourcen zur Erstellung und Wartung für solch eines System im Vordergrund stehen.

Grad	Interaktion
1	Objekte betrachten und rezipieren
2	Multiple Darstellungen betrachten und rezipieren
3	Die Repräsentationsform variieren
4	Den Inhalt der Komponenten modifizieren
5	Das Objekt beziehungsweise den Inhalt der Repräsentation konstruieren
6	Den Gegenstand beziehungsweise den Inhalt der Repräsentation konstruieren und durch manipulierende Handlungen intelligente Rückmeldungen vom System erhalten

Tabelle R-II5-1: Grad der Tätigkeit des Lernenden [2007 SCHULMEISTER]

Da die Wahl für die ITS auf ein kognitives Lernmodell „Lernen durch Einsicht“ und „Aktiver Tutor“ weiter oben in der Doktorarbeit gefallen ist, soll das Konzept geführt und interaktiv sein. Da kommt die konzeptionelle Aufteilung des Lehrmaterials in informatives und interaktives aus dem Kapitel „Aufteilung des Lehrmaterials“ zu Gute, die dafür verwendet werden kann. Interaktives Lehrmaterial wird leicht geführt und ist interaktiv. Einzig und allein interaktives Lehrmaterial reicht allerdings nicht aus, weil informatives Lehrmaterial notwendig ist um nicht beim Start des interaktiven Lehrmaterials selbst das Wissen herauszufinden. Mit dem informativen Lehrmaterial ist die Führung bis zum Start des interaktiven Lehrmaterials genau richtig um nicht konstruktivistisch (zum Beispiel nur ein betreuendes System) zu sein. Damit kann ein ITS ein aktiver Tutor sein. Eine Lehrsitzung wird dann definiert durch erst informatives und dann interaktives Lehrmaterial welche beide aufeinander abgestimmt sind.

III. ARCHITEKTUR

III.1. Softwareengineering

III.1.a. Warum Softwareengineering verwenden?

Ein ITS ist eine komplexe Software, deren Anforderungen schwierig zu erfüllen sind. Die Prinzipien des Softwareengineering erlauben es mit dessen Methoden die Software abstrakt zu spezifizieren, bevor die Programmiertätigkeit begonnen wird. Ziel ist die Verbesserung der Qualität, die Formalisierung und das bessere Verständnis der Software durch Reduzierung der Problemkomplexität. Die Entwicklung muss letztendlich ein Programmierer machen, deshalb sollte seine Sicht berücksichtigt werden. Andere Sichten die noch berücksichtigt werden sollten sind im Kapitel „Beteiligte von ITS-Entwicklungen“ beschrieben. Besonders bei komplexer Software ist Softwareengineering hilfreich, weil nur sehr selten eine einzelne Person das Verständnis des gesamten Systems hat. Es werden viele Personen für die Entwicklung eines ITS benötigt, bei dem jeder nur einen Teil der Arbeit erledigt. Durch Softwareengineering ist es klarer, wie die Aufgaben aufgeteilt werden können.

In dieser Arbeit wird der Teil des Softwareengineerings in den Fokus gerückt, der vor der Entwicklung stattfindet: Software Design. Die Analyse der aussichtsreichen Softwarearchitekturen und wie diese kombiniert werden können, ist ein Ansatz für die Beschreibung von Systemklassen [1994 GARLAN and SHAW]. Es soll kein fertiges Produkt entstehen mit hoher Bedienerfreundlichkeit und sehr wenigen Fehlern in der Programmierung, sondern ein Pre-Standard mit Autorenwerkzeugen für einen großen Teilbereich. Dafür werden die Anforderungen analysiert und das Design der Software in einer Softwarearchitektur erarbeitet. Wenn man eine schon vorhandene Softwarearchitektur nimmt, dann hat man den Vorteil, dass die Spezifikation des Modells schon fertig ist und Fehler wahrscheinlich schon aufgefallen und behoben wurden. Eine bewährte Architektur gibt direkt eine relativ eindeutige Beschreibung, damit für die Entwickler Zeit ist über andere Entscheidungen in der Entwicklung eines ITS zu diskutieren oder auch simpler die Produktionszeit verringert wird. Ein gemeinsames Vokabular und Fachbegriffe verbessern die Kommunikation zwischen den Beteiligten in der Entwicklung, ohne dabei zu sehr ins Detail zu gehen.

III.1.b. Allgemeine Softwarearchitekturen

Es gibt verschiedenen Ideen, um die chaotische Kompliziertheit eines Programms in strukturiert aufgeteilte Komplexität umzuwandeln. Softwarearchitekturen teilen die Verantwortlichkeit des gesamten Programms in mehrere Teile auf, ganz nach dem Motto „Teile und herrsche“. Doch wie genau die Modularisierung und Verkapselung umgesetzt wird, unterscheidet sich teilweise stark. Es gibt bereits bestehende Software- und Systemarchitekturen, welche betrachtet werden sollten bevor eine eigene Architektur entwickelt wird. Denn genauso wie eine ITS-Neuimplementierung das Rad nicht neu erfinden sollte, so sollte sich die Architektur einer Neuimplementierung an allgemeinen bekannten Architekturen anzulehnen, sowohl im Allgemeinen als auch im Speziellen. Hier wird auf die allgemeinen Architekturen [2015 MARK] eingegangen. Die Architekturen sind nicht unter dem gleichen Namen immer exakt gleich, Abweichungen von der vorgestellten Architektur zu einer mit gleichen Namen sollten sich aber im Rahmen halten, sodass es ausreicht eine Vorstellung der Idee zu haben, ohne die genaue Spezifikation zu kennen. Im Folgenden werden unterschiedliche Architekturen betrachtet, welche in den ITS-Architekturen vorkamen. Diese werden unabhängig von schon implementierten ITSs folgend aufgezählt. Die darin erwähnten Klienten sind gleiche Programmteile, ganze Programme oder Systeme, während Komponenten unterschiedliche darstellen. Klienten sind alleine lauffähig, Komponenten müssen es nicht sein.

Peer-To-Peer-Architektur

Bei der „Peer-To-Peer-Architektur sind alle Beteiligten gleichgestellt. Es ist erlaubt dass alle untereinander miteinander kommunizieren, siehe hierzu Abbildung R-III1b-1. Damit entfällt die zentrale Koordination mit mehr Rechten. Das hat eine verbesserte Zuverlässigkeit des Weiterbestehens von Funktionalität, selbst bei Ausfall von zufälligen Teilen der Gesamtarchitektur. Einen simplen sequenziellen Prozess mit klaren Restriktionen darin umzusetzen ist sehr schwierig, weil diese Architektur auf Parallelität aufbaut. Peer to peer wurde für ITSs in Webanwendungen untersucht und empfohlen. Beispiele hierfür sind [1997 BRUSILOVSKY].

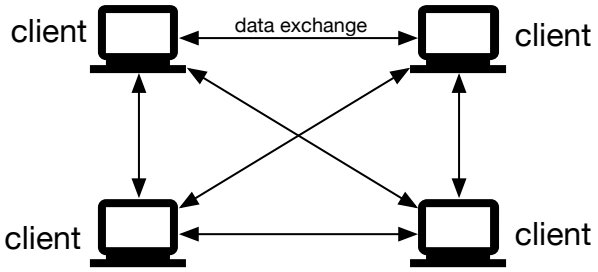


Abbildung R-III1b-1: Beispiel der Peer-To-Peer-Architektur

Client-Server-Architektur

Eine der typischen Webarchitekturen ist die Aufteilung der Software in zwei Bereiche: einen Server, welcher nicht direkt vom Nutzer bedient wird und ein bis mehrere Clients. Ein Server steht bereit für Verbindungen von einem Client. Der Server kann zu mehreren Clients eine Verbindung aufbauen, siehe dazu Abbildung R-III1b-2. Es ist ein verteiltes System bei dem ein Teil nicht lokal vorhanden sein muss. So können mehrere Klienten auf den gleichen Server zugreifen. Die Klienten kriegen dabei immer die aktuelle und kohärente Anwendungslogik. Die Klienten können unabhängig voneinander agieren, benötigen aber immer Zugang zum Server, weil nicht alle Funktionen auf dem Klienten implementiert sind. ITS die darauf aufbauen gibt es viele, einige seien hier genannt [1999 ALPERT, SINGLEY and FAIRWEATHER][1999 SHAW, GANESHAN and JOHNSON][2000 MATSUURA, OGATA and YANO][2000 MITROVIC and SURAWEERA][2000 VIZCAÍNO, CONTRERAS and FAVELA][2002 PRENTZAS, HATZILYGEROUDIS and GAROFALAKIS][2002 YANG and PATEL][2003 CROWLEY and MEDVEDEVA][2003 HARRER][2004 HATZILYGEROUDIS and PRENTZAS][2004 MELIS and SIEKMANN][2005 CROWLEY, TSEYTLIN and JUKIC][2006 LEE, CHO and CHOI][2009b ALEVEN, MCLAREN and SEWALL].

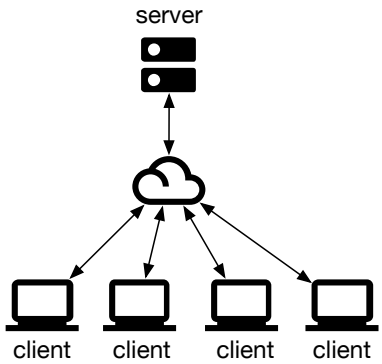


Abbildung R-III1b-2: Beispiel der Client-Server-Architektur

Schichtenarchitektur

Die Schichtenarchitektur (oder auch „layered architecture“) baut darauf auf, dass Programm in mehrere Schichten zu teilen und jeder Schicht einen anderen eigenen Zuständigkeitsbereich zuzuteilen, siehe Abbildung R-III1b-3. Die Anzahl der Schichten wird für ein System festgelegt, muss aber nicht für jedes System gleich sein. Auch der Zuständigkeitsbereich für eine Schicht ist variabel unter den Systemen. Es gibt mehrere Vorschläge für sinnvolle Aufteilungen des Systems in Schichten. Am häufigsten ist die Verwendung der untersten Schicht für die Persistierung der Daten, in zum Beispiel einer Datenbank. Die oberste Schicht verwaltet die Präsentation und Benutzeroberfläche. Bei dieser Architektur dürfen nur direkt aneinander liegende Schichten miteinander kommunizieren. Beispiele für eine deutliche Schichtenarchitektur sind [1992 PUPPE][2000 ZOUAQ, FRASSON and ROUANE][2003 IEEE][2010 GUTIERREZ-SANTOS, MAVRIKIS and MAGOULAS] [2011 MANOUSELIS, DRACHSLER and VUORIKARI].

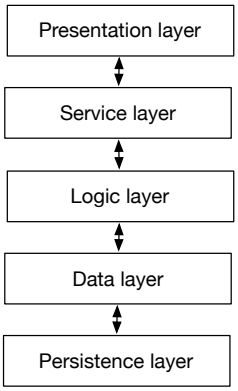


Abbildung R-III1b-3: Beispiel der Schichtenarchitektur

Model-View-Controller-Architektur

Einer der bekanntesten und am meisten eingesetzten Architekturen weltweit für heutige Apps ist die Model-View-Controller Architektur, siehe Abbildung R-III1b-4. Dabei ist durch die viele Verwendung dieser Architektur die Bedeutung unscharf geworden. Apple hat seine eigene Bedeutung von der Architektur, dabei wandelt sie sich in eine reine Layered Architektur mit genau drei Schichten wobei der Controller nur der Mediator ist [2018 APPLE INCORPORATED]. Hierbei sind die Funktionen, welche die Schnittstellen haben dürfen schon vorgegeben, das heißt auch die Verantwortlichkeit jeder Schicht grob festgelegt. Genau diese Architektur hat Microsoft Model-View-Presenter genannt und benutzt selbst die Version, welcher näher an der Ursprungsversion ist. Von dieser Architektur gibt es viele Variationen der Verbindungen untereinander, dennoch ist es in großen Firmen wie Apple und Microsoft eigens definiert [2018 APPLE INCORPORATED][2019 MICROSOFT CORPORATION].

In der Benutzung wie sie der ursprünglichen Idee nahe kommt, gibt die Architektur auch genau drei Typen von Elementen an: „Model“ als den Teil der Architektur der die Daten speichert und den Zugriff darauf ermöglicht, „View“ als den Teil der die Benutzeroberfläche verwirklicht und „Controller“ als ein Element was die Eingaben überprüfen, anpassen, transformieren und letztendlich etwas in das Model abspeichern kann. Die Idee ist hier, dass genau diese drei Teile inhaltlich gut abgrenzbar sind und Änderung in nur einem dieser Teile wahrscheinlich sind, zum Beispiel eine neue Ansicht hinzuzufügen. Die anderen Elemente können dann wiederverwendet werden. Diese Architektur könnte man grob einigen Architekturen zuordnen, wenn man das Studentenmodell und das Domänenwissen zusammenfasst wie bei [1990 NWANA], ansonsten ist ein Beispiel [1985 ANDERSON, BOYLE and REISER] oder [1997 SONG, HAHN and TAK].

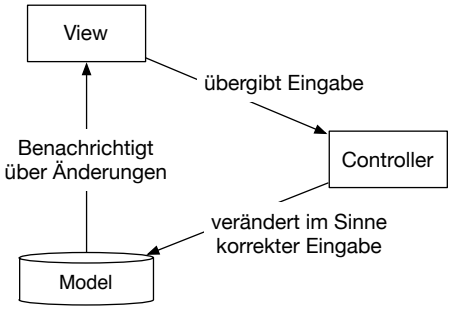


Abbildung R-III1b-4: Beispiel der Model-View-Controller-Architektur

Multitier-Architektur

Die Multitier-Architektur lehnt sich an die Schichtenarchitektur an mit dem gravierenden Unterschied, dass jede Schicht auf einer physikalisch getrennten Hardware ausgeführt wird. Dies wird in Abbildung R-III1b-5 gezeigt, wobei hier der Computer mit Bildschirm zur Interaktion mit dem Nutzer abhängig von anderen Systemen ist. Software für den Zugriff auf Netzwerkressourcen verwendet diese Architektur. Beispiele für ein ITS dieser Art sind [2000 GERTNER and VAN LEHN][2000 ZOUAQ, FRASSON and ROUANE].

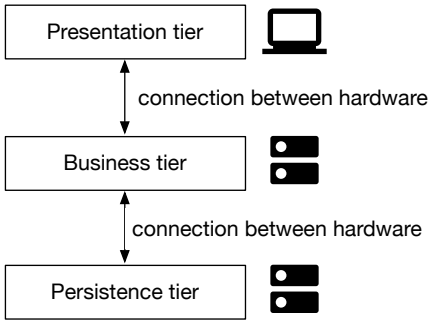


Abbildung R-III1b-5: Beispiel der Multitier-Architektur

Component-Based-Architektur

Bei der Component-Based Architektur geht es darum die Software in einzelne Komponenten aufzuteilen, wobei jede Komponente einen Aufgabenbereich erfüllt. Jede Komponente hat dabei benötigte Eingaben oder Funktionalitäten auf denen sie aufbaut. Von der konkreten Implementation wird dabei zunächst abgesehen, ebenso wie von der Definition von Ausgaben oder Funktionalitäten die sie anbietet, da sie in der Komponente fertig implementiert werden und dabei nicht festgelegt wird wie. Es werden die Schnittstellen der Komponenten selbst, und mit welchen anderen Komponenten sie verbunden sind, definiert. So kann eine Komponente mit einem klar abgesteckten Aufgabenbereich unabhängig von den anderen erzeugt und verändert werden, solange sich ihre Schnittstelle nicht ändert. Die Schnittstellenbeschreibungen sind damit eines der wichtigsten Spezifikationen in dieser Softwarearchitektur. Eine Software mit dieser besteht also aus einer Ansammlung an Komponenten, die mit passenden Komponenten „zusammengesteckt“ werden bis alle gewünschten Funktionen im Programm vorhanden sind, siehe hierzu Abbildung R-III1b-6. Der Vorteil dieser Architektur besteht in der Austauschbarkeit der einzelnen Komponenten. Eine Komponente mit jeweils nur einem Aufgabenbereich erfüllt das „Single-Responsibility-Prinzip“ aus dem SOLID Objekt-Orientiertem-Design [2002 Martin] (das S steht dafür) für besser verständliche Programme. Sehr viele Architekturen von ITS sind komponentenbasiert, einige deutliche Beispiele sind [1984 CLANCEY][1991 ASHLEY and ALEVEN][1994 IKEDA and MIZOGUCHI][2000 MATSUURA, OGATA and YANO][2002 PRENTZAS, HATZILYGEROUDIS and GAROFALAKIS][2003 CROWLEY and MEDVEDEVA][2005 CROWLEY, TSEYTLIN and JUKIC][2011 SOUHAIB, MOHAMED and KADIRI KAMAL EDDINE].

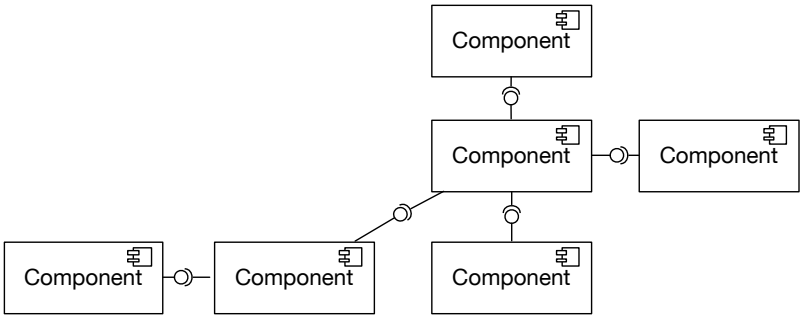


Abbildung R-III1b-6: Beispiel der Component-Based-Architektur

Domain-Driven-Design-Architektur

Die Domain-Driven-Design-Architektur teilt das System in die Domänen auf, welche verwendet werden. Das geht natürlich nur wenn mehrere Domänen vorhanden sind, siehe Abbildung R-III1b-7. Dabei sind oft kleine Domänen gemeint, zum Beispiel die Abbildung in Abteilungen in Firmenstrukturen. Denn anstatt eine aus der Software generierte Struktur vorzugeben, erwartet die Architektur, dass die Infrastruktur, welche man schon von der Realität kennt, auf die Software abgebildet wird. Das hat den Vorteil, dass sich das System vorhandenen Strukturen angleicht und einfacher für Nutzer zu verstehen ist. Dies kann aber auch unnötige Komplexität übernehmen und eine eigentlich softwaretechnisch elegantere Lösung mit einer anderen Struktur ignorieren. Bei sehr starren und langfristig angelegten Firmenstrukturen kann ein Vorteil groß werden. Beispiele hierfür sind [2003 CROWLEY and MEDVEDEVA][1997 SONG, HAHN and TAK].

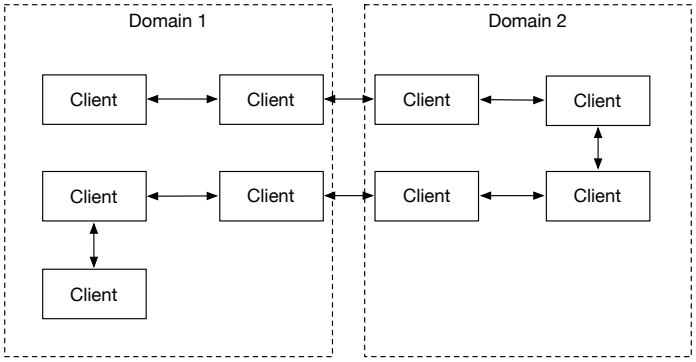


Abbildung R-III1b-7: Beispiel der Domain-Driven-Design-Architektur

Message-Bus-Architektur

Die Architektur mit dem Namen „Message Bus“ wird auch manchmal „Event Bus“ genannt und besitzt einen zentralen Kommunikationskanal, welche die Kommunikation zwischen all den anderen Komponenten übernimmt, siehe hierzu Abbildung R-III1b-8. Dabei wird selten ein Prozess verfolgt oder anderweitig gezielt gesteuert. Die Komponenten kommunizieren indirekt miteinander, indem sie sich anmelden neue Nachrichten von anderen Komponenten zu bekommen. Durch diesen Aufbau ist es möglich neue Komponenten flexibel zu integrieren, ändern oder gar zu löschen ohne andere Komponenten zu ändern oder das Programm gänzlich unbrauchbar zu machen. Die Architektur sorgt aber für eine undurchsichtige Kommunikation der verschiedenen Komponenten und es kann sich damit Dead Code ansammeln. Dead Code ist Code von Komponenten, welche noch auf die Antwort von schon gelöschten Funktionalitäten wartet, die nie ausgeführt wird. Dies kann passieren, weil auch das Löschen von Komponenten durch den Kommunikationskanal verborgen wird. Zudem stellt der zentrale Kommunikationskanal die Kernkomponente dar auf dem alles aufbaut – auch dies kann zu einem Flaschenhals führen. Ein anderer wichtiger Punkt ist die Parallelität der Aufrufe untereinander: der Kommunikationskanal wird bei einer performanteren Lösung nicht synchron arbeiten und dann hat man

Race Conditions zwischen der Kommunikation der verschiedenen Komponenten immer mit zu betrachten. Bei Race Conditions entsteht ein kritischer Wettlauf zwischen zwei Aufrufen von verschiedenen Funktionen bei denen die Reihenfolge wichtig ist. Wenn die Reihenfolge der Aufrufe nur von der Implementierung des Kommunikationskanals abhängt, dann können kleinere Änderungen Fehler erzeugen. Beispiele für diese Architektur sind [1994 IKEDA and MIZOGUCHI][1998 SIEMER and ANGELIDES][2000 HARRER][2000 ROSATELLI, SELF and THIRY][2000 LOS ARCOS, MULLER and FUENTE][2006 OERTEL] [2006 WALKER, KOEDINGER and MCLAREN][2000 ROSATELLI, SELF and THIRY].

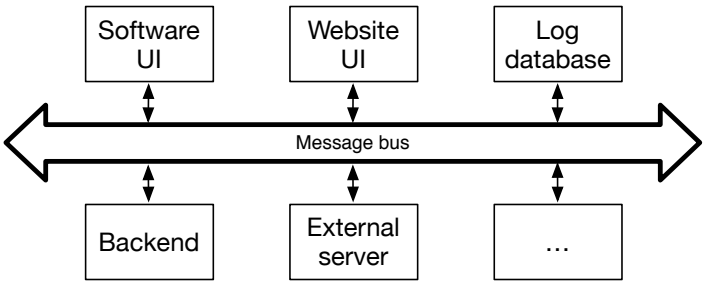


Abbildung R-III1b-8: Beispiel der Message-Bus-Architektur

Repository-Architektur

In der Repository-Architektur geht es darum, dass alle Programme die gleichen Datenlage verwenden, siehe dazu Abbildung R-III1b-9. Die Komponenten oder gleichen Klienten können Daten ändern und das wird direkt auf die anderen Komponenten oder Klienten gespiegelt. Somit sind die Daten unabhängig von den Komponenten und ein Austausch einer Komponente hat keine Änderung der Daten zur Folge. Das führt wie bei der Message-Bus-Architektur zu einem zentralen Element, dessen Performanz und stetige Verfügbarkeit, sowie auch der gleichbleibenden Spezifikation wichtige Elemente zum wartungsarmen Anwenden der Architektur sind. Ein gutes Beispiel ist [2010 CHAKRABORTY, ROY and BASU]. Eine geteilte Datenbank ist auch in der Architektur [2014 GONZALEZ-SANCHEZ, CHAVEZ-ECHEAGARAY and VAN LEHN] vorhanden.

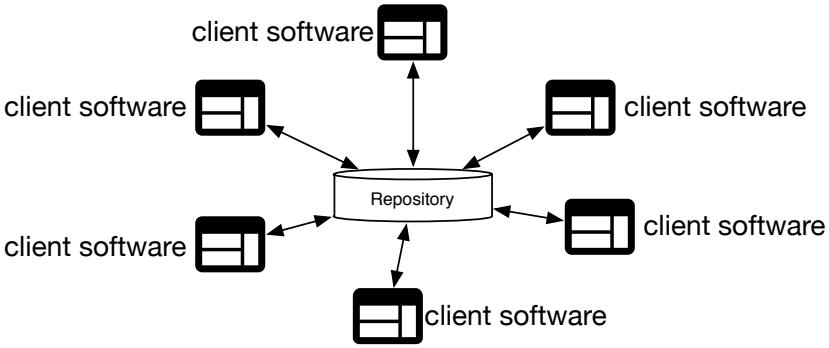


Abbildung R-III1b-9: Beispiel der Repository-Architektur

Pipes-And-Filters-Architektur

Eine sequenziell angeordnete Kette an Funktionalität, welches durch die Eingabe immer einen Output generiert, welcher dann wiederum als Input für die nächste Funktionalität genutzt werden kann, bis am Ende ein Output des gesamten Programms entsteht, siehe dazu Abbildung R-III1b-10. Es integriert einen sehr simplen Prozess direkt in die Architektur, bei dem jede Komponente ein Prozesselement ist. So ist ganz klar definiert welche Kommunikationskanäle für jede Komponente erlaubt sind. Die genaue Spezifikation dieser Architektur erlaubt eine einfachere Anwendung, ihre unflexible Struktur verringert aber das Anwendungsspektrum stark. Ein weiterer Nachteil ist, dass eine Architektur in der ein Sequenzieller Prozess integriert ist keine Parallelität in der Kommunikation zwischen den Komponenten aufweisen kann. Es gibt jedoch erstaunlicherweise genügend Anwendun-

gen bei denen es ausreicht, auch wenn es nicht auf den ersten Blick ersichtlich ist. Der Bezahlvorgang über das Internet ist ein Prozess, welcher aufgrund der Verifizierung sequenziell umgesetzt wurde, bei dem jeder grober Einzelschritt wie eine Blackbox funktioniert. Es geht etwas rein und es kommt danach wieder etwas raus. Solch ein System kann gut in dieser Architektur verwendet werden. Beispiele für diese Architektur sind [1984 O'SHEA, BORNAT and DU BOULAY][1990 NWANA (Self-improving architecture)][2000 FREEDMAN, PENSTEIN and RINGENBERG][2002 HEFFERNAN and KOEDINGER][2010 GUTIERREZ-SANTOS, MAVRIKIS and MAGOULAS] auch wenn mal eine Verbindung nicht exakt in eine Richtung ist.

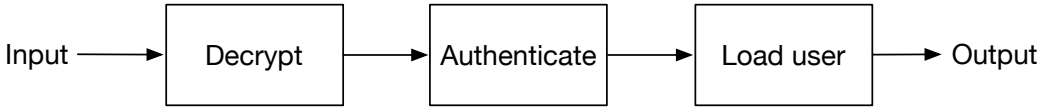


Abbildung R-III1b-10: Beispiel der Pipes-And-Filters-Architektur

Publish-Subscribe-Architektur

Publish-Subscribe ist eine Architektur welche in Webservices häufig Anwendung findet und damit auch in webbasierten ITSs angewendet werden könnte. Da es so viele webbasierte ITSs gibt, wurde eine weitere erfolgreiche Webarchitektur in Betracht gezogen, welche die Client-Server-Architektur ersetzen könnte. Es sollen die Nachrichten von einem angebotenem Service nur an die Komponenten oder Klienten versendet werden, welche auch daran interessiert sind, ohne dabei die Identität der Empfänger zu kennen. Die Klienten oder Komponenten suchen über eine weitere Komponente veröffentlichte Services. An denen können sie sich anmelden und ab da an kommunizieren sie direkt mit dem Anbieter der Funktionalität, siehe hierzu Abbildung R-III1b-11. Es gibt also eine Komponente, die nicht dazu da ist die eigentliche Kommunikation zu kontrollieren, sondern nur um zwei Komponenten oder Klienten zur Laufzeit zur verbinden, um damit angebotene Funktionalität mit denen zu verbinden die danach suchen.

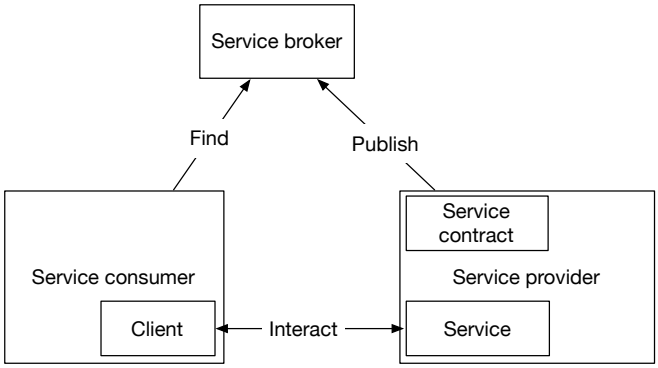


Abbildung R-III1b-11: Beispiel der Publish-Subscribe-Architektur

Zusammenfassung

Zusammenfassend kann gesagt werden dass keine der verschiedenen Architekturen im Allgemeinen schlechter ist als die anderen, sondern jede davon in einem gewissen Einsatzzweck genügend erfolgreich war um als eine gute Lösung darin angesehen zu werden. Gründe für die Wahl einer Architektur können auch Zeitmoden für bestimmte Softwarearchitekturen sein. Jede dieser Architekturen hat Vor- und Nachteile die sich abhängig von der Verwendung auch ändern können. Für die Anwendung einer Architektur im ITS-Bereich sind sie ein guter Einstieg, da verglichen werden kann, was sich in anderen Bereichen des Softwareengineering durchgesetzt hat und man davon bestimmt etwas wiederverwenden kann. Selbst wenn man nicht komplett eine Architektur verwenden möchte so kann man sowohl mehrere Architekturen kombinieren als auch veränderte Formen davon verwenden.

III.1.c. ITS-Softwarearchitekturen

In dieser Doktorarbeit wurden viele Softwarearchitekturen von ITSs gefunden und analysiert [2019b GRAF VON MALOTKY and MARTENS]. Die einzelnen gefundenen ITS-Softwarearchitekturen werden im Anhang in der Tabelle der Architekturen aufgeführt. In den untersuchten Architekturen konnte man fast immer die fünf Grundkomponenten erkennen. Es wird häufig die klassische Architektur mit kleinen oder gar keinen Änderungen verwendet. Nehmen wir zuerst die Gruppe an Architekturen, welche offensichtlich nur aus den Grundkomponenten bestehen: Benutzerschnittstellenkomponente, Lernermodell, Pädagogikmodul und Domänenwissen. Bei dieser Klassifizierung fallen konkrete implementierte Datenbanken-, Hardware- und Benutzeroberflächenartefakte absichtlich weg, weil diese nicht teil einer Softwarearchitektur sind.

Allein schon die Anzahl der gefundenen ITS-Softwarearchitekturen pro Jahr ist dabei interessant, zu sehen in Abbildung A6.3a. Hier ist ein eindeutiger Hype von der Nutzung von Softwarearchitekturen in ITS mit der Jahrtausendwende zu erkennen, danach sind in den geraden Jahreszahlen die Veröffentlichungen höher an in den ungeraden. Dies geht damit einher, dass die meisten Softwarearchitekturen in den großen Konferenzen alle zwei Jahre gezeigt werden, allen voran die „Intelligent Tutoring Systems"-Konferenz. Wenn man diese Statistik mit der von den Veröffentlichungen von ITSs in der Forschung pro Jahr vergleicht, siehe Abbildung R-III1c-1, dann zeigt sich, dass die ITS-Softwarearchitekturen einen Hype nach dem ITS-Hype erfahren haben. Es schein so als ob Wege gesucht wurden, um mit den immer komplexer werdenden ITSs umgehen zu können, es kann allerdings auch ein Hype an Softwareengineering selbst sein.

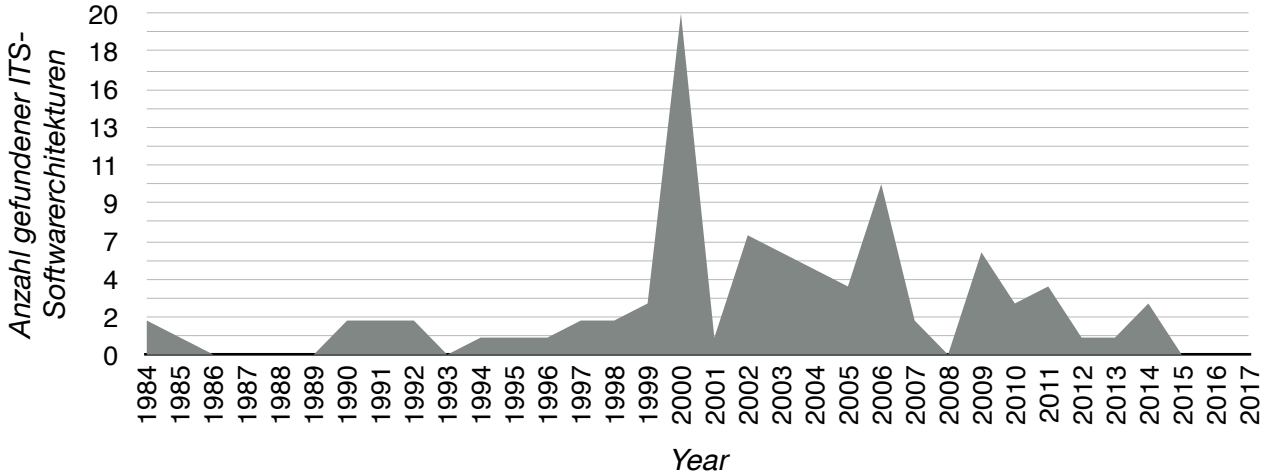


Abbildung R-III1c-1: ITS-Softwarearchitekturen pro Jahr die gefunden wurden

Untersucht man die gefundenen Architekturen danach, wie viele der Standardkomponenten sie beinhalten, so bekommt man folgende Statistik, siehe Abbildung R-III1c-2. Klassische Komponenten beinhaltet wie oben schon beschrieben: Domänenwissen, Pädagogikmodul, Lernermodell und Benutzeroberfläche. Dabei sei beachtet, dass die Funktionalität einer der oben genannten Komponenten manchmal in mehreren Entwurfs- oder Programmierartefakten aufgeteilt werden kann und häufig auch wird. Es ist zu erkennen, dass 54 Prozent aller untersuchten ITS alle klassischen Komponenten abbilden und 82 Prozent mindestens drei der klassischen Komponenten abbilden.

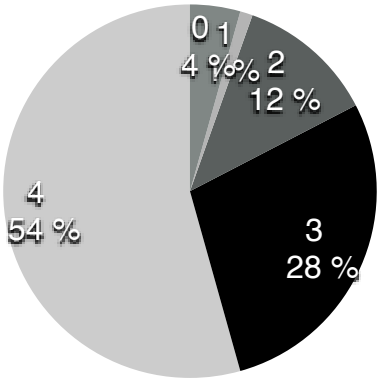


Abbildung R-III1c-2: Prozentuale Anteil der Anzahl der klassischen Komponenten in den gefundenen ITS-Softwarearchitekturen

Es wurde auch untersucht ob bestimmte klassischen Komponenten häufiger implementiert werden als andere. Zu sehen in Abbildung R-III1c-3. Das Domänenwissen ist dabei am häufigsten vertreten. Das Pädagogikmodul als eine Komponente mit sehr vielen Funktionalitäten ist auch noch sehr häufig vertreten. Dabei war aber meist, im Gegensatz zu dem Lernermodell und dem Domänenwissen, nicht ersichtlich wo und ob überhaupt pädagogisches Wissen abgespeichert wird.

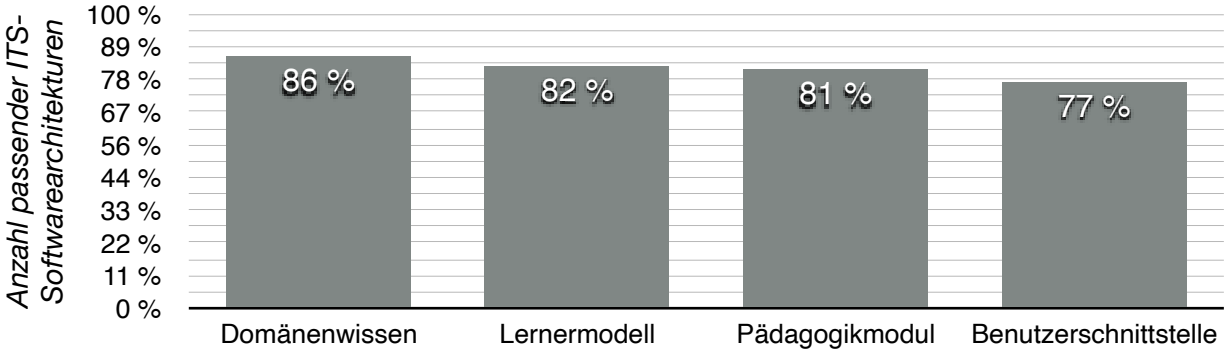


Abbildung R-III1c-3: Prozentuale Implementierung der klassischen Komponenten in den gefundenen Softwarearchitekturen

Übrig bleiben dann noch die Komponenten, welche nicht klassische Aufgaben übernehmen. Diese wurden in zwei Gruppen aufgeteilt: Komponenten mit und ohne Details über eine konkrete Implementierung. Größtes Beispiel hierfür ist es, den Browser und/ oder Server als eigenständige Komponenten in einem ITS zu erwähnen. Die Anzahl der implementationsspezifischen Komponenten variiert stark zwischen null und zwölf und deren prozentuale Verteilung ist in Abbildung R-III1c-4 abgebildet. Die Hälfte der gefundenen Architekturbeschreibungen bleibt allgemein und abstrakt. Bei über einem Zehntel sind es fünf oder mehr Komponenten, die nicht von der Implementierung abstrahiert wurden.

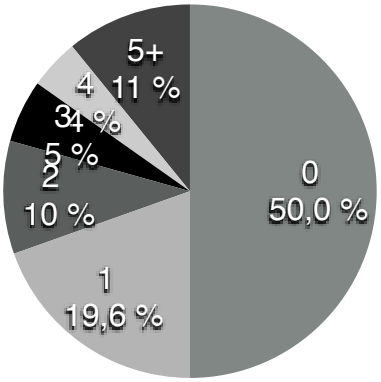


Abbildung R-III1c-4: Anzahl der implementationsspezifischen Komponenten in den gefundenen ITS-Softwarearchitekturen

Auch wenn keine komplett neuen Architekturen gefunden wurden im Vergleich zu der klassischen Architektur, so kann eine Veröffentlichung einer ITS-Architektur dennoch inkrementell innovative Elemente hinzufügen. Der Prozentuale Anteil der Softwarearchitekturen in denen neuen Komponenten hinzugefügt wurden nachdem diese von den implementationsspezifischen Komponenten bereinigt wurden ist in Abbildung R-III1c-5 zu sehen. Dabei zeigt sich, dass nur ein Fünftel aller bereinigten Architekturen neue Komponenten hinzufügt. Der Grundaufbau wird also in den allermeisten Architekturen, wenn überhaupt, nur in implementationsspezifischen Komponenten erweitert.

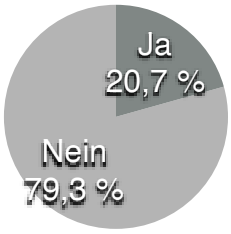


Abbildung R-III1c-5: Neue Komponenten in den gefundenen Architekturen

Ein ITS welches praktisch eingesetzt wird, hätte Vorteile davon, wenn es von Domänenexperten für das Lehrmaterial ohne Fachkenntnisse in der Programmierung neues Domänenwissen dem ITS hinzufügen kann. Ein Autor des Domänenwissens, der selbständig den Umfang des Wissens eines ITS erhöht wäre schneller und günstiger ohne Personen die versuchen zu verstehen was er meinte und es dann in eine Datenbank des ITS zu übertragen. Dabei kann es sowohl sowohl Domänenwissen oder auch Pädagogikwissen sein. Da dies keine kleine Funktionalität darstellt, wäre es sinnvoll es auch in einer Architektur zu repräsentieren. Es wurde daher untersucht, welche der gefundenen ITS-Softwarearchitekturen etwas in diese Richtung repräsentieren, siehe dazu Abbildung R-III1c-6. Hierbei ist zu sehen, dass nur sehr wenige Architekturen Autoren für das Domänenwissen oder Pädagogische Wissen („knowledge engineer“) in der Architektur berücksichtigen.

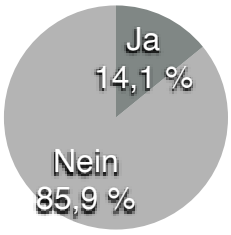


Abbildung R-III1c-6: Berücksichtigung eines Autors für das Domänenwissen oder Pädagogische Wissen („knowledge engineer“) in den gefundenen ITS-Softwarearchitekturen

Architekturbesonderheit Intelligente Agenten

Es ist nicht überraschend, dass es für die in den jeweiligen Jahren in Mode gekommenen allgemeinen Softwarearchitekturen auch ein oder mehrere Pendants in Form von ITS gibt. Eine besondere Aufmerksamkeit in der ITS Community bekam dabei das agentenbasierte Design in Form von Intelligente Agenten. Intelligente Agenten sind Softwareprogramme welche jeweils unabhängig voneinander, meist parallel, Teilprobleme lösen um zum Verhalten des Gesamtsystems beitragen. Eine Entwicklung eines ITS mittels solcher Agenten, bei dem jeder Agent eine Teilaufgabe des Lehrprozesses eigenständig übernimmt und ausführt, scheint sich anzubieten. Die Aufgaben eines Agenten in einem ITS sind vor allem in der Begleitung des Lernenden zu finden. Hier sieht man die Kernaufgaben der klassischen Agenten eins zu eins in der ITS Funktionalität gespiegelt: Einschätzung der Möglichkeiten anhand von Regeln und Vorgaben, sich an einen Plan (hier unter Umständen einen Lehrplan) zu halten sowie bei parallelen Ausführungen von mehreren Aufgaben, zeitkritisch den Auftrag auszuführen. Somit haben agentenbasierte Systeme Regeln, die immer gelten und die Interaktion mit einem Lernenden festlegen und sind besonders vorteilhaft, wenn mehrere Lernende zusammen arbeiten um ein Problem zu lösen. Es gibt ein paar Architekturen, welche versuchen die Vorteile der Agenten zu implementieren [1996 DE BARROS COSTA and PERKUSISCH], [2000 THIBODEAU, BÉLANGER and FRASSON], [2000 LOS ARCOS, MULLER and FUENTE] und [2000 ROSATELLI, SELF and THIRY]. Die agentenbasierte Architektur kam um die Jahrtausendwende in Mode und war theoretisch vielversprechend. Allerdings sank ihre Popularität schnell wieder. Ein Problem, dass dafür sorgte, dass sich solch eine Architektur nicht mittelfristig bei ITS durchgesetzt hat, ist die erhöhte Komplexität eines Agentensystems. Ein Agentensystem wird mit den gleichen Funktionsanforderung komplexer werden, weil die Agenten unabhängig voneinander arbeiten, dass heißt es kann in einem extremen Fall zu einem beliebigen Zeitpunkt eine Meldung von einem Agenten kommen, dass er Daten braucht oder fertig mit der Verarbeitung ist. Der zusätzliche Verwaltungsaufwand von asynchronen Aufrufen an die jeweiligen Agenten muss sich also lohnen und bei einem ITS, welches so schon sehr komplex ist kann dies sehr schwierig werden.

Namen der Komponenten

Die Komponenten wurden alle identifiziert und auch der Name der jeweiligen Komponente wurde notiert. Somit kann festgestellt werden, wie viele unterschiedliche Namen für eine jeweilige Komponente mindestens existieren. Es wurde Groß- und Kleinschreibung missachtet und falls es mehrere Namen in einer Softwarearchitektur für eine Komponente gibt, dann müssen alle Namen schon verwendet worden sein um nicht gezählt zu werden. Das Ergebnis ist in der Abbildung R-III1c-7 zu sehen. Folgend dann alle Namen mit Referenz für die Komponenten. Der unterschiedliche Einsatzzweck des Pädagogikmoduls schlägt sich auch in der stärker differenzierten Benennung nieder, denn es gibt hier die meisten unterschiedlichen Namen dafür. Das Lernermodell hingegen ist in der Benennung am stabilsten. Zur Analyse siehe den Anhang „Tabelle der klassischen Komponentennamen“.

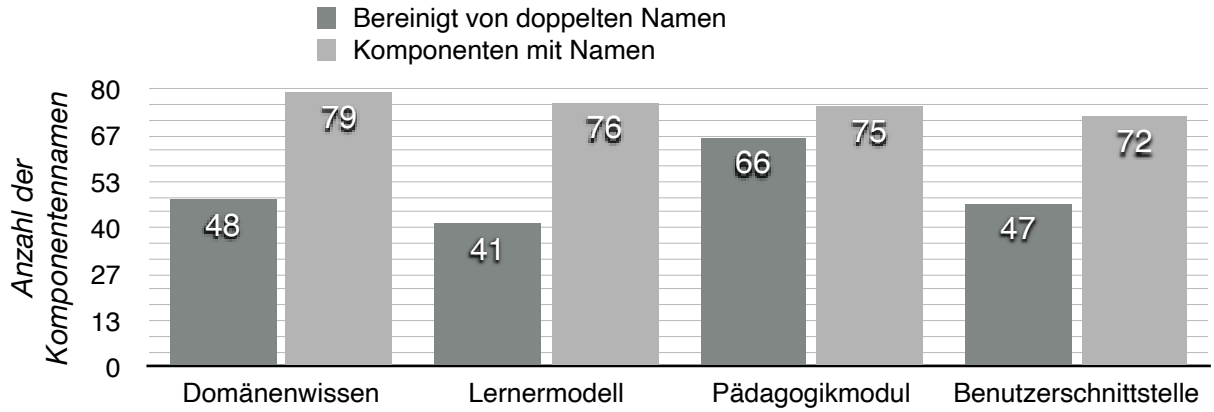


Abbildung R-III1c-7: Anzahl unterschiedlicher Namen für die klassischen Komponenten in den gefundenen ITS-Softwarearchitekturen

III.1.d. Probleme der gefundenen ITS-Softwarearchitekturen

Allgemein gibt es Probleme von denen ITS-Softwarearchitekturen betroffen sind. Erst eine vergleichende Analyse etlicher ITS seit Beginn ihrer Entwicklung ermöglicht eine systematische Aussage über die potentiellen Defizite oder mögliche Entwicklungsstrukturen. Mit 93 untersuchten ITS-Softwarearchitekturen ist dies möglich und die Ergebnisse werden im Folgenden besprochen. Dabei ist zu beachten, dass die Probleme nicht nur bei einem kleinen Teil der Gesamtzahl auftreten und auch dass nicht alle Probleme in einer Architektur vorkommen. Es sind die gesammelten Probleme von den gefundenen Architekturen.

Problem 1: Benennung der Komponenten

Wie man schon am Ende des letzten Abschnitts dieser Arbeit gemerkt hat ist die Benennung der Komponenten nicht eindeutig, sondern sehr unterschiedlich und vielfältig, siehe Tabelle R-III1d-1. Zudem ist die Benennung uneinheitlich in Bezug zu den anderen Komponenten innerhalb einer Architektur. Als Beispiel kann das Domänenwissen unter anderem die Endung „Modul“, „Model“ oder „Knowledge“ haben und dabei hat keiner der anderen Komponenten die gleiche Endung, obwohl sie teils sehr ähnliche Funktionen erfüllen. So gibt es Architekturen welche Komponenten komplett unterschiedlich auslegen und deshalb einen anderen Namen wählen oder es wird unter Umständen nicht auf Einheitlichkeit achten. Teils werden komplett neue Namen für Komponenten verwendet was die Diskrepanz der Benennung erhöht. Das kann davon kommen, dass mehr Aufmerksamkeit auf die in der jeweiligen Forschung entstandene Weiterentwicklung der Komponente gerichtet wird, ohne dabei andere Komponenten mit zu ändern. Der Name einer Komponente ist damit wenig wert, wenn man jedes Mal die Definition genau studieren muss. Ziel sollte es sein keine neue Benennung zu erfinden, sondern vorhandene Komponenten und eine vorhandene weit verbreitete Definition zu nehmen. Weiterentwicklungen sollten nicht in bekannte Komponenten integriert werden, was dann eine Umbenennung erzwingt. Bei gleicher Benennung wäre es einfacher Komponenten wiederzuerkennen. Das ist nicht nur ein Problem von den klassischen Komponenten, sondern auch von anderen hinzugefügten Komponenten, deren Benennung sollte genauso einheitlich benannt werden.

Komponentenaufgabe	Namen
Entscheidungen zum Didaktikverhalten und Pädagogikwissenverwaltung	Tutoring/Tutorial model/component/module
	Pedagogical module/component/expertise/rules
	Instructional expert
	Teaching strategy
	Didaktikkomponente
	Unterrichtsmodul
Kontrolle des Prozesses	Process control/steering
	Central steering component
	System control
	Coordinator
	Steuerprogramm
Benutzeroberfläche Eingaben/ Ausgaben	User/student/author interface
	Kommunikationsmodul
	Problem solving environment
	Presentation engine
	Simulation
Verwaltung des Fortschritts des Lernenden	Student module/model
	User model
	Studentenmodul
	Tutandenmodell
	Learner model/records
Domänenwissensverwaltung	Domain model
	domain knowledge/expert(ise) module
	Expert knowledge/solver
	Course generator
	Expertenmodul
	Domänenexperte

Tabelle R-III1d-1: Inkonsistenz in der Benennung von Komponenten

Problem 2: Bedeutung der klassischen Komponenten wird geändert

Selbst wenn die Benennung einheitlich innerhalb der Architektur ist und sich an Standardbenennung einhält, darf deren Bedeutung natürlich nur minimal von anderen Definitionen abweichen. So könnte jemand der eine Architektur verwendet und schon eine andere kennt, von einem Verhalten der Komponenten alleine durch den Namen auf das Verhalten auf die gleiche Komponente in einer anderen Architektur schließen. Eine Vereinheitlichung der Namen und der Definition könnte auch die Wiederverwendbarkeit der Komponenten erhöhen. Man kann nicht davon ausgehen, dass es nur eine Definition gibt. Allerdings erwähnen die Forschungsarbeiten sehr selten, nach welcher Definition sie sich richten. Anstatt also eine eigene Definition zu erstellen, sollte sich an einen vorhandene Architektur gehalten werden. Eine eigene Definition zu entwickeln bietet sich bei einem derart fortgeschrittenen Systemtyp wie einem ITS nur in zwei Aspekten an: Erstens, wenn es einen triftigen Grund gibt der mehr ist als nur eine kleine Verbesserung eines einzelnen ITS, sondern eine Verbesserung der ITS-Softwarearchitektur im Allgemeinen. Zweitens, wenn man die Softwarearchitektur ganzheitlich betrachtet und sie insgesamt beschreibt und erneuert. Dann reicht es allerdings auch nicht nur aus, etwas zu ändern, es sollte eine allumfassende Analyse geben, warum die neue Softwarearchitektur besser ist.

Problem 3: Nichterwähnung bei der Verwendung klassischer Komponenten anderer Softwarearchitekturen

Architekturen integrieren häufig die klassischen Komponenten: Domänenwissen, Lernermodell, Pädagogikmodul und Benutzeroberfläche. Besonders eigenwillige Benennungen führen dazu, dass nicht schnell klar ist, dass es sich in Wirklichkeit um eine pure Implementierung einer klassischen Komponente handelt. Dies wird aber manchmal nicht

erwähnt, somit wird einem Leser erst später klar, dass es sich eigentlich um eine klassische Komponente handelt. Komplizierter wird es, wenn die Aufgaben einer klassische Komponente einfach nur auf mehrere vermeintlich neue Komponenten aufgeteilt wird oder mehrere klassische Komponenten vereint werden. Im ersten Fall wird die klassische Komponente dann zwar implementiert aber in mehrere Einzelkomponenten aufgeteilt und somit keine neue Funktionalität bereitgestellt. Wird diese pure Aufteilung nicht erwähnt, so kann es wie zwei neue Komponenten wirken. Selten, aber noch problematischer wird es, wenn mehrere klassische Komponenten zusammengeführt werden, besonders wenn dabei noch andere neue Funktionen hinzu kommen. Eine Aufteilung einer klassischen Komponente, ohne dass erkennbar ist, dass es in Wirklichkeit nur eine genauere Beschreibung einer bereits vorhandenen Definition ist, kann zu einer weiteren Spaltung einer bereits etablierten Definition führen. Nur selten gibt es dabei einen Hinweis daraus, dass die einzelnen Komponenten eigentlich die größere Aufgabe einer klassischen Komponente erfüllt. Bei der grafischen Veranschaulichung in der jeweiligen Veröffentlichung wird dies fortgeführt: Die Größe der aufgeteilten Komponenten als grafisches Rechteck ist meist genau so groß wie die anderen klassischen Komponenten. Dadurch werden einzelne Teile einer klassischen Komponente auf die gleiche Stufe gestellt, wie die anderen klassischen Komponenten. Häufig sind alle Komponenten in einer Architektur einfach gleichartig grafisch dargestellt.

Das Problem: Die eigentlichen groben klassischen Komponenten werden also trotz Verwendung gar nicht direkt abgebildet. Eine höhere Detailstufe der klassischen Komponenten ist förderlich, aber nur dann, wenn nicht bekannte Strukturen nur implizit ersichtlich sind und oberflächlich unnötig Eigenkreation entsteht. Es werden damit die klassischen Komponenten versteckt implementiert. Ein Grund dafür könnte sein, dass sich gar nicht erst mit den ITS-Softwarearchitekturen auseinander gesetzt wurde. Ein anderer Grund könnte es sein, dass die vorgestellte Architektur neuer wirken, soll als sie in Wirklichkeit ist indem sie umbenannt wird.

Problem 4: Nichterwähnung bei der Verwendung anderer Softwarearchitekturen

Eine Verwendung einer bereits vorhandenen Softwarearchitektur ist in den meisten Fällen angeraten, um sich Entwicklungszeit zu sparen. Wie in der Analyse in einem vorigen Kapitel zu erkennen, implementieren die große Mehrheit der Architekturen die klassischen Komponenten. Das Problem der Nichterwähnung bei Verwendung tritt nicht nur bei Komponenten auf, sondern auch wenn eine andere Architektur verwendet und leicht verändert wird. Dies geschieht besonders dann, wenn es sich um die klassische Architektur handelt. Selbst wenn die andere Architektur selbst sehr grob und allgemein gehalten wird, fehlt die Zitierung woher diese überhaupt kommt. Wenn es mehrere Softwarearchitekturen gibt, die sich alle ähneln mag es einfacher sein eine Neuentwicklung zu starten, selbst wenn diese fast genauso aussieht wie eine andere. Das verschlimmert leider das Problem der Undurchsichtigkeit und Nützlichkeit von ITS Softwarearchitekturen. Hier schafft die Entwicklung einer Referenzarchitektur voraussichtlich Abhilfe.

Problem 5: Keine komplette Integrierung einer anderen Softwarearchitektur

Aufbauend darauf, dass die implementierte Komponenten oder ganze Softwarearchitekturen nicht erwähnt werden, gibt es ein weiteres Problem. Man verändert die passend aussehende Architektur nach seinen Wünschen und lässt dabei Teile weg, die unnötig oder zu aufwändig erscheinen. Das mag bei einer hoch spezialisierten Architektur funktionieren, aber eine allgemeine ITS-Softwarearchitektur mit fünf Komponenten ist dermaßen auf seine Kernfunktionalität heruntergebrochen und abstrahiert, dass ein Weglassen einer Komponente schon dazu führen kann, dass es kein ITS mehr ist.

Problem 6: Begründung der Entscheidungen fehlt

Geht man von sinnvollen Änderungen an einer Softwarearchitektur aus zur Verbesserung im Allgemeinen, dann reicht es nicht, die neue Architektur zu präsentieren. Viele der gefundenen Architekturen begründen leider nicht warum sie etwas ändern mussten, sondern zählen nur die Vorteile auf. Die vorliegende Analyse zeigte, dass diese sogenannten Vorteile oft rein akademischer Natur sind und auch ohne Änderung einer anderen Architektur implementiert werden könnten. Jede neue Architektur die allgemeingültig für ITS verwendet werden soll sollte begründen, warum sie so aufgebaut ist wie sie ist und wenn sie nur in einzelnen Teilen abweicht von vorherigen Architekturen, dann warum genau eben jenes geändert worden ist. Wenn jemand jetzt nach einer ITS-Softwarearchitektur sucht, ist es schwer herauszufinden warum eine besser ist als die andere.

Problem 7: Unabstrahierte Komponenten

Dass Beschreibungen von ITS-Softwarearchitekturen viel zu implementierungsspezifisch sind kommen auch häufig vor. Entscheidungen in der Implementierung, welche über die Architektur hinaus gehen und bei denen eine Wiederverwendbarkeit schwer ist, werden gleichgestellt mit abstrakten Komponenten. Es wird nicht nur die Domäne des ITS sondern auch das genaue Anwendungsgebiet, mit in der Architektur verwoben beschrieben. Als Beispiel sei hier die Architektur von [2003 CROWLEY and MEDVEDEVA] genannt, siehe dazu Abbildung R-III1d-2.

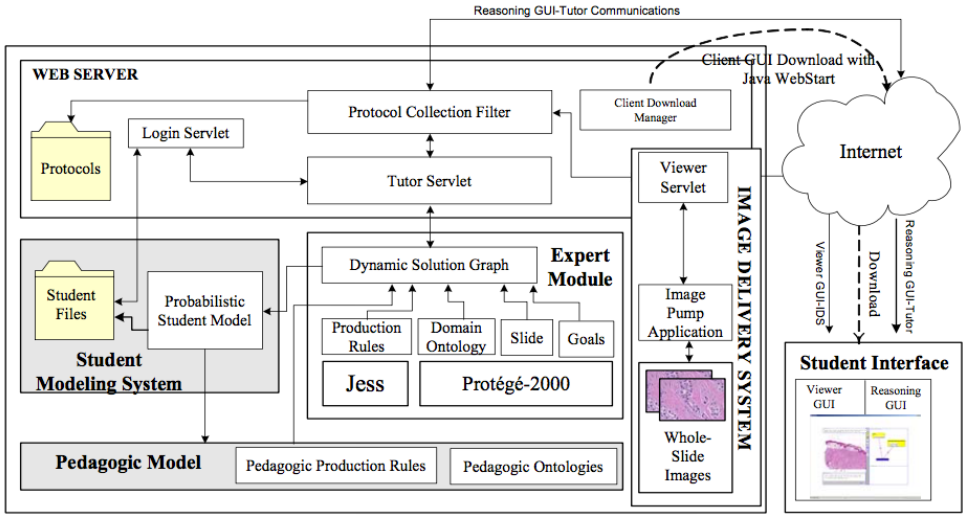


Abbildung R-III1d-2: Architektur welche viel zu implementierungsspezifisch ist [2003 CROWLEY and MEDVEDEVA]

Ein Grund hierfür könnte sein, dass die Architekturen mit diesem Problem erst in einer Adhocentwicklung ohne Softwareengineering implementiert wurden und erst nachträglich wurden die Architekturen aus bereits bestehenden Systemen erstellt. Solch ein Weg macht es schwieriger wieder zu abstrahieren. Ein anderer Grund könnte sein, dass die Personen, welche die Architektur erstellt haben nur ihre implementierungsspezifische Architektur vorstellen ohne dabei großen Wert auf die Wiederverwendbarkeit davon zu legen. Damit ist der Abstraktionsgrad und damit auch die damit verbundene Arbeit geringer, weil die konkrete Implementierung und die Architektur näher beieinander liegen. Manchmal wird auch einfach nur die klassische Softwarearchitektur genommen und dort eine unabstrahierte Komponente hinzugefügt um eine neue Architektur zu erstellen [1998 ELSHEIKH and STICKLEN]. Das ist die Kombination der falschen Abstraktionsebenen und ist nicht zielführend.

Problem 8: Pädagogikmodul zu ungenau

Ein besonders großes Problem sowohl mit der Benennung als auch mit der Bedeutung hat eine der vier klassischen Komponenten: Das Pädagogikmodul. Zwar variieren die Funktionsweisen und Definitionen der anderen Komponenten auch, aber weniger und in einem viel kleineren Rahmen als bei dem Pädagogikmodul. Wie schon im vorherigen Kapitel gezeigt, hat es die meisten unterschiedlichen Namen. Das Pädagogikmodul wird sehr häufig gar nicht als eine Wissenskomponente gesehen, wie das Lernermodell und das Domänenwissen. Auch wenn manchmal die anderen beiden Wissenskomponenten nicht nur für den Wissenszugriff zuständig sind, sondern auch weitere Funktionalitäten haben, so sind sie fast immer auch eine Sammlung an Daten. Besonders auffällig ist, dass dies beim Pädagogikmodul fehlt — es beinhaltet bei der Mehrheit der Architekturen viel mehr als nur den Wissenszugriff auf das pädagogische Wissen. Größtenteils ist bei diesen Architekturen gar nicht ersichtlich, dass es sich um eine Datensammlung handelt, sondern es wird die Funktionalität beschrieben, welche implizit dann pädagogisches Wissen benötigt. Das Pädagogikmodul ist damit in der praktischen Umsetzung in den gefundenen Softwarearchitekturen eine Komponente mit viel Funktionalität und eines davon ist die Abspeicherung des pädagogischen Wissens. Die pädagogischen Entscheidungen des Systems mit so einer Komponente sind damit intransparent und sind nicht einfach in einer eigenen Komponente zugänglich. Die Entscheidungen des Systems sind damit auch nicht einfach nachzuvollziehen. Ohne einfachen Zugriff auf das pädagogische Wissen ist auch eine Autorenschnittstelle schwieriger und damit von einem Didaktiker ohne gute Kenntnisse des Systems nicht änderbar. Generell ist eine Autorenschnittstelle bei didaktischen Wissen in den Architekturen extrem selten.

Selbst die Bedeutung der gefundenen Funktionalitäten des Pädagogikmoduls ohne die Speicherung des pädagogischen Wissens zu betrachten ist sehr inkohärent. Folgend aufgelistet welche Funktionalitäten in den Pädagogikmodulen der untersuchten ITS-Softwarearchitekturen gefunden wurden.

- Vorgefertigte Fehlerdatenbank: Eine Datenbank welche eine Auflistung von Fehlern von Lernenden hat um damit den optimalen Umgang mit einem Lernenden direkt nachdem diesen Fehler begangen wurde
- Vorgefertigte Antwortendatenbank: Eine Datenbank an fertigen Antworten die ein ITS geben kann um damit per se schon überprüfbare und damit im Zweifel fehlerfreiere Antworten zu besitzen
- Vorgefertigte Lernpfadedatenbank: Eine gute Sequenzialisierung des Lehrstoffs ist ein wichtiger Bestandteil von guter Lehre. Wenn die Lehreinheiten bekannt sind, können auch effektive vorgefertigte Pfade durch den Lehrstoff gespeichert werden
- Profilgenerierung: Eine Profilerstellung als zusammenfassende Auswertung der Historie eines Lernenden um Werte zu bekommen die so nur umständlich direkt aus der Historie ersichtlich sind, wie zum Beispiel die Vorlieben und Schwächen eines Lernenden bezogen auf Themen und Tagen
- Direkte Analyse der Eingaben des Lernenden: Eine direkte Analyse der Eingaben ist bei einem diaglogbasierten ITS häufiger im Einsatz, welche nur die letzte Aktion des Lernenden berücksichtigt
- Generierung der Antworten: Eine Generierung von kompletten Antworten von Grund auf passend zum Lernenden
- Angepasste Antworten: Die Anpassung von vorgefertigten Antworten an das Profil des Lernenden mit einer etwaigem Verwaltung der vorherigen Dialoge zur Dialogfortführung
- Wann aktive Hilfe gezeigt wird: Wenn es eine aktive Hilfe gibt, dann muss das ITS entscheiden, wann dies erfolgen soll. Ob zum Beispiel nach einer gewissen Zeit-

dauer, wenn eine bestimmte Anzahl an Fehlern gemacht wurde oder ein vergleichsweise grober Fehler gemacht wurde [2004 MARTENS]

- Angepasste Hilfe: Anpassung der angezeigten Hilfe (aktive oder passive Hilfe) an den Lernenden, zum Beispiel des Anzeigen von der richtigen Menge Inhaltstext oder Hervorhebung bestimmter Stellen in der Darstellung
- Empfehlung des Lehrstoffs: Empfehlung von Lehreinheiten welche als nächstes gelernt werden sollen, weil sie zum Wissensstand des Lernenden passt
- Anpassung des Lehrstoffs: Die Lehreinheiten müssen nicht starr sein, sondern können auch an die Bedürfnisse des Lernenden noch verändert werden, um zum Beispiel den Detailgrad an den Wissensstand des Lernenden anzupassen
- Generierung von Lehrstoff: Lehreinheiten könnten auch komplett generiert werden, dies kommt bei Regelbasierten Domänen vor wie Physik und Mathematik, bei dem Übungen aus einfachen Gleichungen mit einer fehlenden Variable sein können
- Didaktikprozessverwaltung: Da einem ITS eigentlich auch ein didaktischer Prozess zugrunde liegt, kann dies auch in diesem Modul realisiert werden, welche Optionen dem Lernenden als nächstes zur Verfügung gestellt werden um damit zu entscheiden was als nächstes gemacht wird

Es sind insgesamt zu viele Funktionalitäten die in einer Komponente vorkommen. Generell wird das Thema „Lehren“ von den Funktionalitäten nicht verfehlt, aber wenn jeder etwas anderes darunter versteht ist das schädlich für eine Softwarearchitektur, welche sich als abstrahierten Standard durchsetzen will. Zusätzlich zur Verwirrung trägt bei, dass das pädagogische Wissen (als englische Version „pedagogical knowledge“) auch häufig der Name einer Komponente (Pädagogikmodul) ist und dies damit gleichzeitig der Name für zwei Dinge ist: Eine Wissensart und eine Komponente. Die historisch spätere Einführung dieser Komponente im Vergleich zu den anderen klassischen Komponenten erhöht die Missverständnisse des Pädagogikmoduls noch zusätzlich.

Problem 9: Informelle Beschreibung

Eine exakte Beschreibung der Aufgaben und Schnittstellen der Komponenten kann durch eine formale Beschreibung erreicht werden. Doch leider sind die ITS-Softwarearchitekturen nicht formal beschrieben. Harrer und Martens erkannten dieses Problem bereits im Jahr 2006 und versuchten, eine Mustersprache für ITS zu entwickeln [2006 HARRER and MARTENS]. Leider wurden diese Bestrebungen damals nicht weitergeführt. Die vorhandenen gefundenen Softwarearchitekturen kann man nur mit viel Aufwand formalisieren. Die ungenaue Spezifikation und die unterschiedliche Auslegung was eine Komponente überhaupt machen soll tragen dazu bei.

Einer der wichtigsten Veranschaulichungen für eine Softwarearchitektur ist die visuelle Darstellung. Deshalb wurden im Anhang auch alle gefundenen Architekturen mit ihrer visuellen Repräsentation und Zitierung aufgelistet. Da es bei den Grafiken der gefundenen ITS-Softwarerchitekturen fast ausschließlich undefinierte grafische Elemente gibt, die sich nicht an formale Visualisierungsstandards halten, ist nicht eindeutig was die Pfeile, Boxen und anderen Zeichnungen genau zu bedeuten haben. Die Genauigkeit der visuellen Präsentation korreliert meist mit der Präzision des zugehörigen Textes. Es gibt in den aufgelisteten Architekturen keine textuellen Beschreibungen die formeller sind als ihr visuelles Pendant. Es wirkt so, als ob die viele Grafiken intuitiv aus einem Prosatext entstanden sind, nicht aber aus einer genauen formalen Spezifikation, welche sich in eine formale visuelle Repräsentation transformieren lässt, wie zum Beispiel in die Unified Modeling Language (UML) [2018 OBJECT MANAGEMENT GROUP (OMG)]. Dies ist insbesondere für informatorische Beschreibungen nicht zielführend und nicht wissenschaftlich adäquat.

Informelle Beschreibungen sind weniger nützlich in der Entwicklung und Analyse von ITS, weil sie einen viel größeren Interpretationsspielraum erlauben. Am Beispiel von Abbildung R-III1d-3 sieht man zwar generell was gemeint ist, aber es ist nicht klar was ge-

nau gemeint ist. Es ist nur schwer erkennbar was sich der Autor mit den Farben und den unterschiedlichen Pfeilen gedacht haben könnte. Die Veröffentlichung selbst geht auf eine mögliche grafische Notation nicht ein — so dass davon ausgegangen werden muss, dass es sich um eine kreative Eigenleistung handelt. Grafische Darstellungen wie diese sind häufig, siehe ausgefallene Beispiele [1995 VASANDANI and GOVINDARAJ][2000 LOS ARCOS, MULLER and FUENTE][2000 UNIVERSITÄT HANNOVER][2002 GAMBOA and FRED][2006 LEE, CHO and CHOI][2007 FUNDA and KADIR][2012 LIN, WANG and CHAO]. Die Genauigkeit steckt im konkreten ITS, diese ist aber kein Modell, sondern eine von Entscheidungen über die Implementierung belastete Umsetzung. Dies hat zur Konsequenz, dass das Modell nicht weiter genutzt werden kann und damit als singulärer Test eines ITS keine wissenschaftliche Weiterentwicklung ermöglicht. Forschung kann auf diese Weise nicht erfolgen.

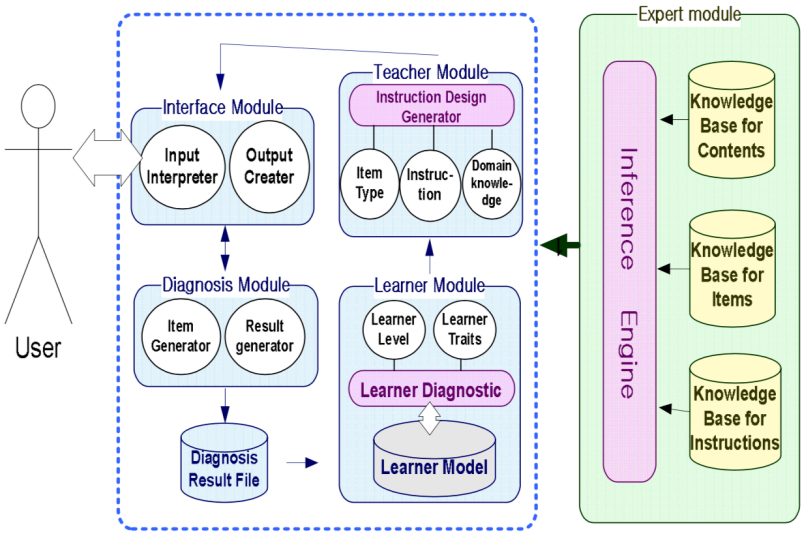


Abbildung R-III1d-3: Nichtformelle Visualisierung einer ITS-Architektur [2006 LEE, CHO and CHOI]

Es gibt implizite Regeln, größtenteils nicht aufgeschrieben oder formalisiert. Dennoch wissen neue ITS-Entwickler diese Regeln nicht, obwohl diese es besonders benötigen. Ohne den Formalismus führt eine Mehrdeutigkeit einer Regel zu willkürlichen Interpretationen, die falsch sein können. Ohne Erfahrung hat es eine erhöhte Wahrscheinlichkeit fehlerbehaftet zu sein. Selbst wenn es fehlerfrei wäre, so entspricht die Interpretation nicht der eines anderen Entwicklers. Somit entstehen zwischen ITS oder auch innerhalb eines ITS Konflikte. Eine Architektur zu formalisieren und sich an die Spezifikation zu halten benutzt vordefinierte Elemente und erzeugt eine allgemeingültigere Beschreibung. Wiederverwendung dieser Struktur erzeugt eine Vergleichbarkeit und ein sofortiges Verständnis der Definition der Elemente, wenn man die Architektur selbst kennt. Schwer zu begreifende, nichtoffensichtliche Probleme werden verhindert und das Verhalten ist vorhersehbarer. Die Grafiken und Prosatexte sind nicht genau genug, bieten aber abstrakt gesehen einen Anhaltspunkt. Wenn die Grafiken aber nicht genau erklärt werden und die Texte eher schwammig formuliert werden, dann wird eine Architektur stellenweise schnell anders als vorgesehen aufgefasst und die Architektur ist dann nur noch Ideengeber anstelle Vorgabe einer klaren Spezifikation. Eine erhöhte Fehlerrate und undefiniertes Verhalten die aufgrund von einer abweichenden Verwendung als die vom Autor gewollte, können genauso entstehen, wie eine sinkende Kohärenz mit anderen ITS die auch die gleiche Softwarearchitektur verwenden.

Problem 10: Verbindungen vernachlässigt

Es gibt zwei Probleme mit den Verbindungen die gemeinsam als eine Vernachlässigung bezeichnet werden können. Einmal werden die Schnittstellen nicht genügend beschrieben

und dann gibt es noch die zu wenige Einschränkung der Bindung zwischen den Komponenten. Die Grundlage um das erste Problem zu verstehen ist: Die Komponenten der Softwarearchitekturen für ITS sollen unabhängig voneinander entwickelbar sein, nur die Schnittstelle sollte festlegen, welche Funktionen diese Komponente erfüllen muss ohne zu spezifizieren wie sie das umsetzt. Wenn man also für die Komponenten nur beschreibt was sie tun sollen ohne genauer zu spezifizieren wie die Schnittstellen beschaffen sind, so wird es schwieriger die Komponenten auszutauschen. Die Grundlage die man für das zweite Problem verstehen muss ist: Eine Kopplung in der Softwareentwicklung ist die Abhängigkeit von zwei verschiedenen Softwarekomponenten oder Systemen zueinander. Wenn die Kopplung unter den Komponenten hoch ist, dann ist es schwieriger diese auszuwechseln, weil mehr an anderen Komponenten geändert werden muss, schlimmstenfalls sogar etwas an grundlegenden Funktionsweise einer Komponente, welches dann schnell sehr aufwändig wird. Um eine Entkopplung zu gewährleisten hilft die Datenkapselung der Komponenten, bei der die interne Datenstruktur der Komponente verborgen wird. Der Zugriff auf deren Funktionen erfolgt ausschließlich über eine festgelegte Schnittstelle. Die Schnittstelle besteht aus einkommenden und ausgehenden Daten, deren genaue formale Spezifikation notwendig ist um zu entscheiden ob eine Komponente mit einer anderen ausgetauscht werden kann. Die Verknüpfungen der Komponenten untereinander wird in den allgemein einsetzbaren ITS-Architekturen meist vernachlässigt modelliert. Ein Kommunikationskanal von einer beliebigen Komponente zu jeder anderen ist möglich. Die einzelnen Verbindungen zwischen den Komponenten ist nicht genügend beschrieben oder können häufig nicht korrekt vom Diagram abgelesen werden. Bei vielen gefundenen Architekturen besitzen die Komponenten zu viele Aufgaben gleichzeitig. Dies führt unter Anderem dazu, dass es viele Verbindungen zwischen allen möglichen Komponenten gibt. Optimal wären so wenig wie nötig Verbindungen zwischen den Komponenten zu haben und damit nur die zu behalten die wirklich sinnvoll sind. Wenn jede Komponente mit jeder anderen kommunizieren kann, ist das mit der Kopplung von Code vergleichbar, die klare Zuständigkeiten zwischen den Komponenten schafft und damit das „Single-Responsibility-Prinzip“ aus SOLID [2002 Martin] erfüllt. Damit entsteht ein klare Begrenzung der Verantwortungsbereich für eine Komponente. Ein Negativbeispiel ist wenn die Benutzeroberfläche direkt von allen möglichen Komponenten angesprochen werden kann. Das Problem ist sowohl die schwammige Beschreibung der einzelnen Komponenten als auch eine leger und offen gestaltete Spezifikation der Verknüpfungen der Komponenten. Dies führt zu Komponenten mit mehren Aufgabenbereichen und ungenau abgesteckten Grenzen, bei denen nicht klar, was die Komponente nicht machen darf, sondern nur was sie mindestens machen sollte. Beispiel an der Abbildung R-III1d-4, bei dem die Linien mit dem X in der Mitte die Verknüpfungen zwischen den Komponenten darstellt. Es sind damit Pfeile in beide Richtungen. Der Nutzer interagiert mit dem System aber es ist unklar, welche Komponente ist dafür zuständig (als erstes womöglich das „interface“) ist. „Problem data“ hat gar keine Verknüpfung liegt aber auf der Komponente „A casebase“.

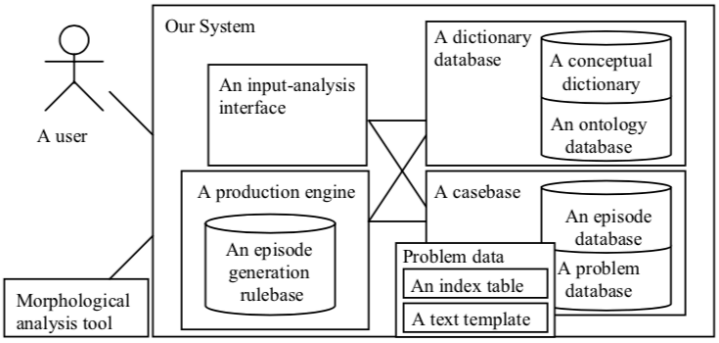


Abbildung R-III1d-4: Schlechte Verbindungen zwischen den Komponenten [2006 KOJIMA and MIWA]

Wenn man die Architektur abstrahiert um deren Problem 10 auf den Kern zu visualisieren, welche leider bis heute immer wieder genau so „neu erfunden“ wird, sieht es so wie in Abbildung R-III1d-5 aus.

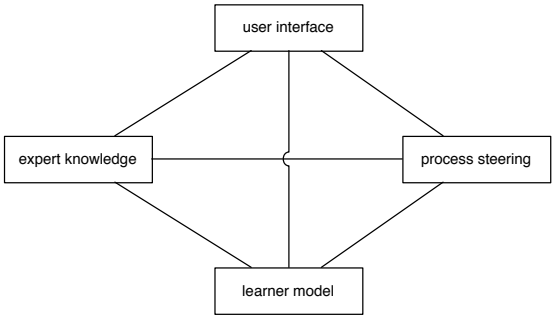


Abbildung R-III1d-5: Abstraktion von Architekturen mit schlechten Verknüpfungen

An dieser Abbildung kann man auch besser die Problematik der Verknüpfungen erkennen. Jede Komponente ist mit jeder Verbunden, weil jede Komponente ein Teil des Prozesses steuert. Kein Prozess und keine zentrale Verwaltung des Geschehens ist abgebildet. Rein vom Modell her könnte eine Komponente jederzeit einspringen und die Benutzeroberfläche kapern um etwas neues anzuzeigen. Für Softwareentwickler - insbesondre für Teams, in denen eine geteilte Aufgabenbearbeitung erfolgt — bleibt völlig unklar, welche Komponente welche Prozesskompetenz haben soll. Generell gilt: Die Verknüpfungen der Komponenten sind das einziges Mittel der Kommunikation zwischen den Komponenten. Eine genauere Spezifikation der Verknüpfungen führt automatisch zu einer besseren Definition der Beschränkungen. Es gibt in allen untersuchten Architekturen zu wenig Spezifikation darüber was die Komponenten eigentlich dürfen oder nicht, größtenteils würde eine genauere Betrachtung der vorhandenen Verknüpfungen zu mehr Einschränkungen führen.

Problem 11: Prozess in Architektur

Eine Prozessbeschreibung ist ein weiteres Mittel um die Verwendung der Softwarearchitekturen genau zu beschreiben und eine gute Idee um zusätzliche Spezifikationen neben der Architektur zu erschaffen. Per Definition ist ein Prozess nicht teil einer Softwarearchitektur. Trotzdem passiert es, dass ITS Entwickler versuchen, Architektur und Prozess gemeinsam darzustellen, anstatt den Prozess eigenständig zu beschreiben. Damit wird er selten noch einer Softwarearchitektur gerecht. Interessant ist, dass dieser Fehler sogar bei der Entwicklung der „Learning Technology System Architecture“ (LTSA) Standard von der IEEE gemacht wurde. Die wenig zielführende Vermischung aus Prozess und Architektur führte dazu, dass der Standard von der IEEE nach wenigen Jahren wieder zurückgezogen wurde [2018 IEEE STANDARDS ASSOCIATION], siehe dazu Abbildung R-III1d-6. Abbildung R-II4a-3 hat in der linken Hälfte auch eine stark sequentielle Verbindung zwischen den größeren Komponenten, welche auf eine Integration eines Prozesses hindeuten, der auf dieser Abstraktionsebene schnell hinderlich werden kann. Auch wenn die Architektur nicht ausschließlich für ITSs gültig sein soll. Es verwendet die YOURDON-Notation [1975 YOURDON], welche eigentlich Datenflussprozesse ausgelegt ist, dabei allerdings immer von einer Architektur redet. Dabei lässt sich der Lerner nicht von dem System unterscheiden. Die Beschreibung der Aufgabenbereiche ist gelungen — dies war auch die Hauptaufgabe der LTSA. Der Prozess zieht sich durch drei Komponenten, wobei jeweils zwei Komponenten immer mit einer von zwei Datenbanken kommunizieren dürfen. Jede der Komponenten hat eine direkte Verbindung zum Lernenden. Es ist sehr allgemein gehalten und bietet nicht genügend Beschreibung für eine Neuimplementierung. In einem Paper [2002 MOEBUS, ALBERS and HARTMANN] wurde die Architektur als fehlerhaft beschrieben, wenn sie zur Umsetzung eines ITS angewandt wird. Dennoch hat die IEEE mit

der Bestrebung verdeutlicht, dass es eine wissenschaftliche Formalisierung wichtig ist, um die ITS Entwicklung entlang einer standardisierten Architekturbeschreibung zu vereinen. Die vorliegende Dissertation liefert hier einen entscheidenden Beitrag.

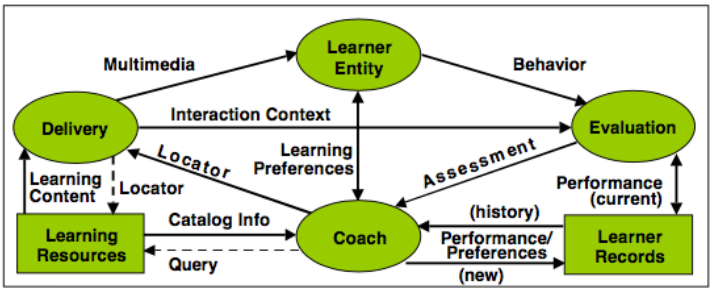


Abbildung R-III1d-6: Schicht 3 der LTSA, IEEE 1484.1 [2003 IEEE]

Problem 12: Keine Softwarearchitektur

Besonders schwer zu implementieren sind Beschreibungen von ITS, welche im Kern der Darstellung nicht als Architekturen bezeichnet werden können. Es scheint so, als wollen die Entwickler dieser Systeme eine Idee vermitteln, und spiegeln dazu nur einen kleinen Teil des Systems wieder. Als Beispiel sei das Modell von [1998 SELF] genannt, siehe hierzu Abbildung R-III1d-7. Da auch hier der Anschein erweckt wird, dass es sich um ein programmiertes Softwareartefakt handelt, kann der Eindruck entstehen, dass es hier aus der Idee heraus „Programming is an art“ [2007 KNUTH] entwickelt wurde und keine Richtlinien des Softwareengineering zugrundegelegt wurden. Derartige Systeme sind zur Nachnutzung gänzlich ungeeignet und massiv abhängig von der singulären Implementierung. Der wissenschaftliche Nutzen kann kontrovers diskutiert werden.

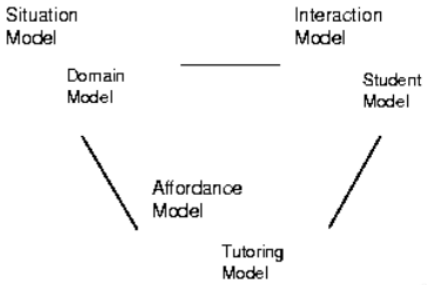


Abbildung R-III1d-7: Keine echte Architektur [1998 SELF]

Problem 13: Keine Weiterentwicklung

Viele Systeme entwickeln die klassische Architektur nicht weiter und implementieren die klassische Architektur, beschreiben dann ihre Systemarchitektur ohne diese formal richtig darzustellen. Und so bleibt wie bei [2013 AHUJA and SILLE] eine Architektur bei der man nicht genau weiß welche Aufgaben die einzelnen Komponenten haben, siehe Abbildung R-III1d-8. Statt dessen sieht man nur das Abbild der klassischen Architektur in unterschiedlicher Darstellung der immer gleichen Elemente in einer von wenigen Anordnungen.

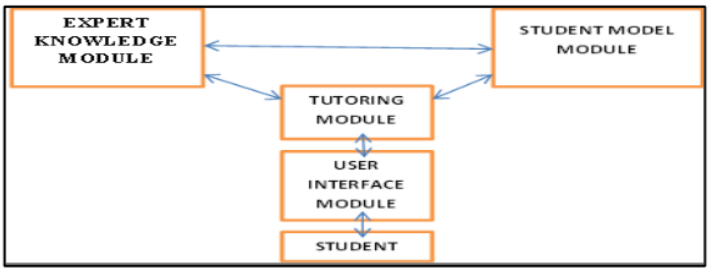


Abbildung R-III1d-8: Anwendung einer klassischen Architektur [2013 AHUJA and SILLE]

Zusammenfassung der Probleme

Die Qualität eines ITS wird häufig von einem Experten des Domänenwissens angepriesen. Dieser testet das System auf seine inhaltliche Korrektheit stichprobenartig. Bei einem komplexen und dynamischen System wie ein ITS ist dies sehr schwer und kann zu ungenügenden Tests führen. Die Qualität umfasst mehrere Faktoren wie die Lehrqualität, die Qualität des zugrundeliegenden Wissen, die Qualität der Software selbst (woran messbar und die Qualität des Modells.

Das generelle Problem ist schon die Verfügbarkeit von Autoren und deren unterschiedliche Arten das Wissen syntaktisch zu speichern. Die große Anforderung an die Experten macht ein ITS zusätzlich sehr aufwändig, besondern wenn das Wissen noch nicht formal genug für ein Computerprogramm gespeichert wurde. Die Arbeit des Experten sollte demnach so einfach wie möglich sein. Das kann mit einer einheitlichen Struktur und Prozess mit festgelegten Zugriffsschnittstellen für das in dem System gespeicherte Wissen erreicht werden. Eine Vereinheitlichung welches die proprietären Systeme nie haben werden.

Eine andere Methode die Qualität eines ITS zu messen ist durch den Wissenszuwachs und subjektive Einschätzung ihrer Lernerfahrung von Probanden. Eine Evaluation mit den Lernenden ist aufwändig und benötigt eigentlich eine doppelte Blindstudie mit bildungswissenschaftliche und empirischen Qualitätskriterien. Doch der Wissenszuwachs ist bei unterschiedlichen Systemen mit unterschiedlicher sichtbarer Benutzerführung und unsichtbarer interner Struktur schwierig als einziges Vergleichsmaß zu nutzen, wenn so viele Faktoren Einfluss haben könnten. Denn auch wenn die Probanden nicht bewusst begreifen was ihren Lernfortschritt beeinflusst und ob und wie sie gerade lernen, so kann nicht ausgeschlossen werden, dass es ein Erfolg den falschen Effekte zugeordnet erden. Zum Beispiel beeinflusst womöglich das Aussehen der Oberfläche des ITS mehr als erwartet den Lernenden. Zusätzlich werden häufig werden keine Standardtests verwendet, weil sie ungeeignet für ITS sind und immer wieder neue für das jeweilige ITS angepasste Tests schneiden sehr gut ab [2016 KULIK and FLETCHER].

Zusammenfassend können die Hauptprobleme in den folgenden Punkten zusammengefasst werden:

- Eine immerwährende Weiterentwicklung eines ITS ohne sich an eine Vorgabe zu halten führt zu unterschiedlichen ITSs bei dem die Übertragbarkeit von Daten, ein Autorensystem für mehrere ITSs und deren Vergleichbarkeit nicht gegeben ist
- Fortwährende Weiterentwicklung von ITSs ohne Referenzstandard für die Architektur. Hier ist zu beobachten, dass die grundlegenden Ideen der Architektur in den letzten Jahren nicht mehr verändert wurde. Die Vielfalt der Veröffentlichungen täuscht darüber hinweg, dass es im Bereich der Kernarchitektur kein wirkliche wissenschaftliche Innovation mehr gegeben hat. Eine Referenzarchitektur existiert trotzdem nicht
- Die Entwicklung von Autorensystemen für ITSs ist quasi unmöglich, weil es keine Referenzarchitekturen gibt. Ohne Autorensystem ist der Systemtyp ITS aber nicht weiter nutzbar und bleibt auf der Ebene der singulär nutzbaren Forschungsprototypen hängen
- Die Qualität eines ITS wird durch unterschiedliche Experten, in Didaktik, Softwareentwicklung und dem Domänenwissen bewertet. Die wiederum ein ITS unter Umständen alle unterschiedliche Funktionen als wichtig erachten
- Lernpsychologische und bildungswissenschaftliche Untersuchungen ignorieren andere Eigenschaften eines ITS, die auch zum Beispiel einen Wissenszuwachs beim Lernenden erwirkt haben könnten. Sei es die Benutzeroberfläche, das Verhalten oder andere Eigenschaften, welche indirekt auch durch die Architektur d

- ITS-Softwarerchitekturen sind entweder fast nur die klassische Architektur und zu allgemein gehalten oder zu spezifisch für eine Domäne/Implementierung, es scheinen dömanenspezifische oder implementationsspezifische Details in der Architektur durch
- Weiterentwicklungen von ITS-Architekturen überarbeiten diese nicht, sondern fügen nur ein kleines neues Element hinzu, es ist keine systematische Verbesserung die eine echte wissenschaftliche Weiterentwicklung bedeuten würde. Zudem gilt für die untersuchten Architekturen:
    - Keine bis wenig Einschränkungen der Verbindungen zwischen den Komponenten in einer Architektur
    - Benennung und Aufgabenzweck einer Komponente ist nicht klar spezifiziert und unterscheidet sich häufig leicht von den klassischen Komponenten für die Standardkomponenten
    - Es gibt keine formale Beschreibung eines ITS im Ganzen. Keine explizite Einigung auf formales Modell
- In der Geschichte der ITS-Entwicklung gibt es viele Adhoc-Implementierung durch neue Erkenntnisse im Bereich der künstlichen Intelligenz oder aus dem Bereich der Psychology
- Keine explizite Einigung auf formales Modell
    - Implizite Verwendung von bestimmten Eigenschaften welche nicht im klassischen Referenzmodell erwähnt werden
- Es wird sich nicht an das gehalten was eine Softwarearchitektur ist, weil entweder ein Prozess hinzugefügt oder die Mindestanforderungen nicht eingehalten werden
- Das größte Problem bei den klassischen Komponenten hat das Pädagogikmodul mit viel zu vielen Bedeutungen
- Unübersichtliche Beschreibungen von ITS, Referenzierungen von genutzten Architekturen und Begründungen was warum geändert wurde fehlen

Nicht nur die Anzahl sondern auch die tiefgreifende Art der Probleme zeigt auf dass etwas getan werden muss. ITSs haben sich schneller entwickelt als deren Softwarearchitektur. Die aktuellen ITS-Architekturen werden oft von dem bereits implementierten ITS generiert und fokussieren sich nur darauf dieses eine zu implementieren. Die gefundenen ITSs sind häufig Insellösungen mit Adhoc-Implementationen relativ unabhängig von dem Stand der Forschung. Die allgemeineren ITS-Architekturen sind veraltet oder zu trivial. Das macht sich besonders dann bemerkbar, wenn die Beschreibung unzureichend ist. Die Komponenten werden häufig ausführlicher beschrieben als die Verbindung untereinander.

III.1.e. Vorschläge für die klassische Softwarearchitektur

Nach der Analyse der Zustände von vorhandenen ITS-Softwarearchitekturen und der Diskrepanz zu den tatsächlichen eingesetzten Softwarearchitekturen, wurden folgende Vorschläge erarbeitet, welche die klassische ITS-Softwarearchitektur verbessern.

Vorschlag 1: Bezeichnung/Bedeutung der klassischen Komponenten vereinfachen

Viele Probleme beruhen auf einem falschem Verständnis oder falschen Umsetzung der klassischen Komponenten. Da die Benennung der Komponenten sich evolutionär entwickelt hat, ist sie undurchsichtig und uneinheitlich, wie weiter oben in der Statistik zu erkennen ist. Die Komponenten welche das Wissen abkapseln werden häufig auch im Redegebrauch einfach als Synonym für das Wissen verwendet.

Damit wird das Wissen, welches hauptsächlich im Gebrauch des ITS steckt, unabhängig von den Komponenten folgendermaßen genannt.

- Domänenwissen
- Lernendenwissen
- Pädagogikwissen

Diese Namen sind eindeutig, einheitlich und simpel. Es wird das Wort „Lernender“ als Stamm genommen, weil laut Duden „Lerner“ sich mehr auf das Lernen einer Sprache bezieht, siehe folgende Definitionen.

- Lerner: „jemand, der (eine Sprache) lernt“ [2018a DUDENVERLAG]
- Lernender: „jemand, der etwas lernt“ [2018b DUDENVERLAG]

Es wird eine vereinfachte einheitliche Benennung aller bisherigen Komponenten vorgeschlagen mit ganz klaren vereinfachtem Einsatzzweck, indem ihnen komplexere weiterführende Verarbeitungsaufgaben entzogen werden. Die Funktionalität umfasst nur die Speicherung, Herausgabe, Editierung oder einfache Transformation des Wissens ohne eine Weiterverarbeitung oder Analyse. Um keine Verwirrung zwischen dem gespeicherten Wissen und der Komponente zu schaffen, wird den Komponenten zusätzlich das Suffix „-komponente“ hinzugefügt. Daraus ergibt sich ein Schema für eine Namenskonvention. Das Schema ist durch die folgenden Regeln definiert.

- Die Namen von Komponenten werden mit dem Suffix „komponente“ benannt
- Erster Wortbestandteil klassifiziert die Nutzungskategorie: Lerner, Domäne, Pädagogik
- Mittlerer Wortbestandteil gibt Zusatzinformation (Handelt es sich um „Wissen“, das im ITS abgebildet werden muss, so wird dies im mittleren Wortteil abgebildet, zum Beispiel „Domänenwissenskomponente“)

Als Beispiel ist das Lernendenwissen, das gespeicherte Wissen und die Lernendenwissenkomponente die Komponente, die sich um die Speicherung dessen kümmert. Was das Wissen alles beinhaltet kann noch immer sehr komplex innerhalb der Komponente modelliert werden, nur nicht mehr was weiterführend damit gemacht wird. Da Software produziert werden soll, muss es ein wenig größer gefasst werden um von den eigentlich gespeicherten Daten zu abstrahieren: Die Speicherung, Editierung und der Zugriff auf das Wissen mit Umwandlung in eine möglichst ähnliche implementationsunspezifische abstrahierte Variante wird in den Wissenskomponenten des ITS abgebildet.

Die Komponente für Gestaltung der visuellen Benutzeroberfläche und anderen Schnittstellen der Kommunikation zwischen ITS und Lernendem hat bereits einen Namen, der sehr weit verbreiteten Namen „User Interface“ / „Benutzeroberfläche“. Wie auch bei den Wissenskomponenten wird das Suffix „-komponente“ hinzugefügt. Dennoch wird die Komponente zur Benutzerschnittstellenkomponente umbenannt, weil ihre Aufgabe mehr umfasst, als ihre bisherige Hauptaufgabe die Benutzeroberfläche zu verwalten. Auch die Benutzerschnittstellenkomponente unterliegt dann der neuen Beschränkung und darf keine anderen Funktionen als die Anzeige gegebenen Lehrmaterials und die Rückmeldung der Interaktion mit Eingabeelementen auf der Anzeige besitzen. Es darf das Lehrmaterial in keiner Weise verändern, sondern sich nur selbst daran anpassen. Als Beispiel, darf die Benutzeroberfläche zwar bestimmen wo und wieviel gleichzeitig angezeigt wird, aber muss es auf eine passende Art und Weise immer ermöglichen den kompletten Lehrstoff zu betrachten. Ein Weglassen und damit Zensieren an Wissen ist für die Benutzerschnittstellenkomponente nicht erlaubt. Dabei fehlen Funktionen die nicht passen, wie auditive Schnittstellen (Mikrofon und Lautsprecher) und andere Ein- oder Ausgabegeräte verwalten.

Damit entstehen die folgenden Namen für die klassischen Komponenten die natürlich auch mit englischer Übersetzung daherkommen:

- Domänenwissenkomponente / Domain knowledge component
- Lernendenwissenkomponente / Learner knowledge component
- Pädagogikwissenkomponente / Pedagogical knowledge component
- Benutzerschnittstellenkomponente / User interaction component

Dies ist nur ein Vorschlag für die Benennung der klassischen Komponenten und deren Funktion, aber natürlich noch keine Abbildung eines ITS. Zusätzliche Funktionalität darf nicht in eine dieser Komponenten integriert werden und sollte zum größten Teil sowieso nicht diesen klassischen Komponenten zugeschrieben werden und kann daher auch eigene Namen tragen, welche sich bei Möglichkeit an diese anlehnen soll. Die Möglichkeiten eines ITS mit einer klaren rigorosen Definition der klassischen Komponenten macht es unmöglich, viele der aktuellen ITS nur mit den klassischen Komponenten zu kreieren. Wenn die klassischen Komponenten in ihrer Funktionalität strikt klein gehalten werden, insbesondere die Pädagogikwissen-Komponente, dann können diese vier Komponenten nicht alle Komponenten sein, die ein ITS laut Definition dieser Arbeit braucht. Aber es verhindert, dass diese Komponenten für Funktionalitäten missbraucht werden, für die sie nie gedacht waren. Statt dessen werden Softwarearchitekten gezwungen diese Funktionalitäten explizit zu machen. Das erhöht das Verständnis, die Sichtbarkeit und die Angreifbarkeit (auf die Untersuchung des Zwecks) von neuen Funktionalitäten.

Vorschlag 2: Hinzufügen der Funktionalität der Prozesssteuerungskomponente

Über ein Drittel der gefundenen ITS-Softwarearchitekturen besitzt klar genau eine zentrale Komponente, welche nicht in der klassischen Architektur abgebildet ist, siehe dazu die Auswertung der gefundenen ITS-Softwarearchitekturen in Abbildung R-III1e-1. Der dafür passende Name wird hier definiert als „Prozesssteuerungskomponente“. Die Prozesssteuerung (in Englisch process steering) ist ein wichtiger Teil in einer Lehrsoftware. Eine ausführliche Analyse der Notwendigkeit solch einer Komponente gibt es in [2004 MARTENS]. Da es eine notwenige Funktionalität ist, die in der Mehrheit nicht in einer Komponente repräsentiert ist, wird sie häufig implizit implementiert. Dies hat den Nachteil, dass sie weder einsehbar und überprüfbar, noch austauschbar und verbesserungsfähig ist. Auch wird häufig nicht erkannt, dass der Lehrprozess ein signifikanter Bestandteil eines Lehrvorgangs ist.

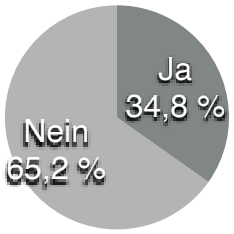


Abbildung R-III1e-1: Hat eine Prozesssteuerungskomponente in den gefundenen ITS-Softwarearchitekturen

Die Prozesssteuerungskomponente entscheidet was als nächstes im System gemacht wird. Während die anderen Komponenten auf einen bestimmten Bereich spezialisiert sind um eine Aufgabe zu bewältigen, so bildet die Prozesssteuerungskomponente den grundlegenden Prozess ab mit dem das System abläuft. Sie ist die zentrale Kontrolle anderer Komponenten, übernimmt dabei aber nicht die Entscheidungsgewalt über die einzelnen Bereiche oder überprüft deren Ergebnisse, sondern überlässt das der jeweilig passenden Komponente. Ihre Aufgabe ist es den nächsten groben Schritt aus dem Prozess herauszusuchen und zu wissen welche die richtige Komponente für die Bewältigung dafür ist. Damit verknüpft es die spezialisierten anderen Komponenten miteinander indem es die

Ergebnisse einer Komponente als Eingaben für ein oder mehrere andere Komponenten verwendet.

Wenn man sich die Komponenten nimmt, welche von [1987 Wenger] aufgezählt und beschrieben wurden und welche bis heute Bestand hat so merkt man, dass die Prozesssteuerungskomponente nicht vorkommt. Zudem wird auch ein ITS-Lehrprozess in Forschungsarbeiten beschrieben [2004 MARTENS]. Die Prozesssteuerungskomponente kommt in den untersuchten ITS-Softwarearchitekturen wenig zum Einsatz oder wird nicht erwähnt, obwohl ihre Funktionalität unabdinglich ist. Dies kann mehrere Gründe haben, die folgend aufgezählt werden.

- Sie ist in den einzelnen anderen Komponente jeweils ein wenig integriert, die Verantwortung ist damit nicht an einer Stelle gesammelt oder abgekapselt
- Die Verantwortlichkeit dieser Komponente wird teilweise gar nicht zu einem ITS zugeschrieben sondern nur zu einer konkreten Umsetzung und gehört damit „nur“ zur normalen Implementierung eines Programms. Diese Ansicht ist aber nicht zu Ende gedacht, weil dann würde auch die Benutzerschnittstelle keine ITS-Komponente sein
- Bei den vorhandenen Architekturen gar kein formaler Prozess zu Grunde liegt, aus dem einfachen Grund, dass keiner bewusst entwickelt wurde.
- Der Prozess ist von minderer Qualität und soll daher wenig präsent sein. Er wird sozusagen versteckt.

Fazit

Diese beiden Vorschläge verbessern zwar die klassische Architektur, dennoch nehmen sie aber keine genügend zufriedenstellende Änderung an der klassischen Architektur vor, als dass diese als ein neuer Standard genommen werden kann.

Die vorhandenen Komponenten zu definieren mit der Reduzierung des Aufgabenbereichs der Komponenten und somit das Setzen von einheitlichen Funktionalitäten für ähnliche Komponenten macht deutlich, dass die Komponenten viel mehr Funktionalität besaßen die nicht klar kommuniziert wird. Diese Funktionalität muss herausgearbeitet und kategorisiert werden. Das neue Benennungsschema wird dabei helfen neue Komponenten Einheitlich zu benennen.

Die Einführung einer neuen Komponente, welche den Prozessablauf reguliert und schon in einem signifikanten Anteil der gefundenen Architekturen vorhanden ist zeigt insbesondere, dass nicht nur Funktionalitäten in den einzelnen Komponenten versteckt waren, sondern sich auch über alle Komponenten in einem ITS verteilen können. Es wird daher mit Abbildung R-III1e-2 nur ein unzureichende Version einer verbesserten klassische Architektur gezeigt mit der notwendigen Option zur Erweiterung durch zusätzliche Komponenten an der Prozesssteuerungskomponente.

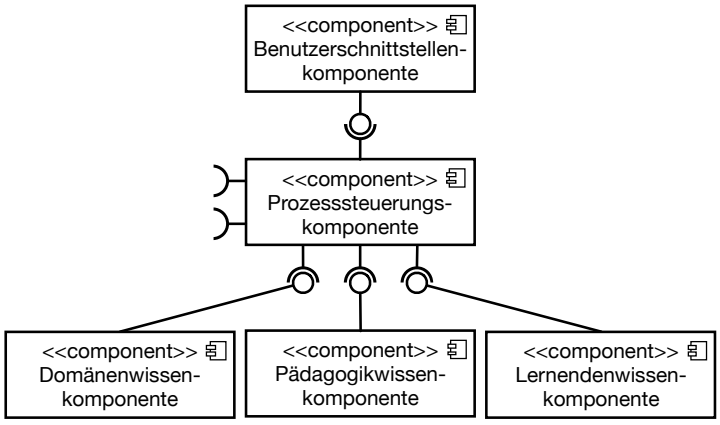


Abbildung R-III1e-2: Komponenten der klassische ITS-Softwarearchitektur in einem Komponentenmodell und zusätzlicher Prozesssteuerungskomponente

Aufgrund der vielen Probleme mit den vorhandenen ITS-Softwarearchitekturen und des hohen Alters der klassischen Architektur ist es problematisch diese weiterzuentwickeln. Kleine Änderungen werden nicht reichen um die gewünschten neuen Funktionalitäten zu integrieren. Eine komplette Neuentwicklung einer allgemein gültigen ITS-Softwarearchitektur wäre die beste Möglichkeit ITSs zu vereinheitlichen. Dafür benötigt es allerdings nicht nur das Wissen und die Grundlagen von ITS und die Analyse von vorhandenen ITS und deren historische Entwicklung wie es in bisher in der Doktorarbeit beschrieben wurde, sondern eine Aufarbeitung der Grundlage eines neuen Konzepts auf Basis der Anforderungen im heutigen Unterricht.

III.1.f. Wahl einer Softwaregrundlage

Wie in den Nachteilen der ITS schon erwähnt sind ITS spezialisierte Systeme, an deren Fokussierung in einem Gebiet die Wiederverwendbarkeit eines ITS leidet. Abstraktion wird durch die Konzeption eines Modells einer Softwarearchitektur und eines Prozesses geboten. Um diese Abstraktion wieder in ein konkretes ITS umzuwandeln ist viel Arbeit notwendig. Diese Arbeit für ITS-Entwickler auf Softwareebene zu verringern ist Aufgabe eines Softwareframeworks. Vorhandene Softwareframeworks für ITS werden so gut wie gar nicht verwendet, obwohl sie große Vorteile bieten in der Entwicklungsgeschwindigkeit und in dem Gesamtaufwand. Einer der Probleme von vorhandenen Softwareframeworks ist, dass sie keinem formalem Modell folgen. Ein Softwareframework könnte ein Modell durchsetzen und automatisch zu einer Vereinheitlichung führen. Dennoch ist ein Softwareframework selbst eine Konkretisierung, welche nicht in allen Fällen optimal sein kann und auch an Softwareverfall („software regression“) leidet, bei dem durch bestimmte Ereignisse etwas nicht mehr so funktioniert wie vorgesehen. Häufig geschieht dies durch Updates von abhängiger Software (wie das Betriebssystem, andere Softwareframeworks, Netzwerkressourcen und so weiter) oder Hardware. Ein Softwareframework und das dazugehörige Wissen muss für die Aufrechterhaltung der vollen Funktionalität sich mit seinen Abhängigkeiten zusammen weiterentwickeln. Einem Softwareverfall kann wie bei realen Maschinen nur mit ständiger Wartung und Weiterentwicklung entgegen gewirkt werden. Deshalb ist es unerlässlich, dass die eigentliche Idee unabhängig von einem Softwareframework entwickelt und formalisiert wird und es sich an die Prinzipien des Softwareengineering zu halten, um sie auch in einem anderen Softwareframework umsetzen zu können.

Die Systemarchitektur betrachtet die verfügbaren Ressourcen und den Anwendungsfall, da sind Dinge relevant, wie die Skalierbarkeit des Gesamtsystems. Weil für eine breite Anwendung aber der generische Teil der Lösung gesucht wird, wird daher die Softwarearchitektur anstatt einer Systemarchitektur gewählt. In dieser Arbeit wird eine Systemarchitektur weder analysiert, noch entwickelt. Das zu entwickelnde Lehrsystem wird absichtlich nicht die Hardware mit deren Aufgaben- und Lastverteilung oder den verwendeten Dienste konkretisieren. Es wurden verschiedene allgemeine Softwarearchitekturen betrachtet. Es wird aussortiert welche Architekturen ungeeignet sind für den Zweck eine weitestgehend allgemeingültige Architektur zu erstellen. Dabei fällt eine, wegen ihrer Konkretheit, von vorn herein aus: Domain-Diven-Design-Architektur. Die Multitier-Architektur ist aufgrund ihrer Nähe zur Hardware unpassend.

Die Message-Bus-Architektur wurde schon in einem ITS-Softwareframework als Architektur verwendet mit dem Namen „JaBInt“ [2006 OERTEL], siehe Beschreibung der ITS-Softwareframeworks. Die Probleme dieses Frameworks beruhen dabei auf seiner Architektur, die Kommunikation ist nicht von vorn herein beschränkt. Der Event-Bus ist hier die zentrale Kommunikationseinheit und müsste dafür Sorge tragen, dass nur bestimmte Kommunikation zwischen Komponenten erlaubt ist. Dies bildet automatisch einen Flaschenhals, weil jede Überprüfung Systemleistung kostet und damit nur die maximale

Kommunikation zwischen den Komponenten beschränkt wird auf die Leistungsfähigkeit einer Komponente. Ein Problem welches auch bei einer Prozesssteuerungskomponente in der ersten Idee einer Architektur, wie in den Vorschlägen erwähnt, auftreten kann und in dem zu entwickelnder Softwarearchitektur verhindert werden sollte. Die Message-Bus-Architektur lässt die Frage der Beschränkungen der Kommunikation komplett unbeantwortet und der Aufwand wird auf eine zu implementierende Komponente abgewälzt, anstatt es wenigstens teilweise in der Architektur zu spezifizieren.

Die Peer-To-Peer-Architektur hat seine Einsatzzwecke in den verteilten Systemen. Eine sinnvolle Umsetzung von ITSs auf dieser Architektur würde zum Beispiel ein Agentensystem erfordern, bei dem dann ein oder mehrere Agenten auf jeweils einem Klienten laufen. ITSs mit Agentensystemen haben erschwerte Bedingungen durch die Entwicklung der Parallelisierung und der damit einhergehenden Komplexität. Dafür gewonnene Vorteile (wie die horizontale Skalierung für erhöhte Performanz).

Die Pipes-And-Filters-Architektur ist sequenziell aufgebaut, der Ablauf des Prozesses ist deterministisch und beinhaltet wenig Komplexität. Für ein ITS ist dies wegen der geforderten Adaptivität nicht geeignet. Der Prozess darf nicht komplett starr von der Architektur vordefiniert sein.

Die Repository-Architektur besitzt ein für ITS für die Lehrforschung interessantes Konzept. Es wird eine zentraler Datenspeicher verwendet. Das ermöglicht es die Ergebnisse von allen laufenden Klienten immer am einem Ort verfügbar zu haben. Da aber ein ITS sehr viele, auch persönliche Daten sammelt um gut zu funktionieren, ist es schwierig die Repository-Architektur als Grundarchitektur zu nehmen. Denn dann wird davon ausgegangen, dass viele der persönlichen Daten des Lernenden in jedem Fall in das zentrale Repository hochgeladen werden muss. Es sollte nicht selbstverständlich sein, dass es ein Einverständnis für die Datenübermittlung vom Lernenden gibt. Besonders die Verbindung vom Klienten zu dem Repository müsste stabil sein. Die Vorteile für den Softwareklienten sind dabei kurzfristig gesehen kaum vorhanden. Solch eine Architektur verkompliziert ein ITS im Kern nur unnötig. Die Idee der Sammlung der Daten zur späteren Auswertung für Forschungszwecke ist eines der Ziele für die zu gestaltende Architektur, aber nicht innerhalb eines ITS sondern über alle ITSs hinweg.

Die Client-Server-Architektur ist in ihren Einsatzzweck zu eingeschränkt. Es soll nicht von vornherein eine Teilung des Systems in zwei zusammenarbeitende Systeme geben, dies wäre bei einem Einsatz im Bereich von Webtechnologien sinnvoll, doch bei einem ganzheitlichen System ungeeignet. Da das Konzept unabhängig von der Implementierung sein muss, kann diese Architektur nicht der beste Ansatz sein. Zwei Aspekte sind trotzdem auch für die im folgenden vorgenommene Entwicklung relevant und werden dort berücksichtig: Erstens gibt es eine einfache Aufteilung zwischen der Benutzeroberfläche und dem restlichen ITS. Zweitens ist es möglich, dass mehrere Nutzer gleichzeitig auf das System zugreifen.

Die Publish-Subscribe-Architektur ist eine weitere in den Webtechnologien häufig verwendete Architektur. Das Trennen von einer Dienstsuche und dem Anbieter ergibt im ITS-Kontext wenig Sinn, weil die Dienste von dem ITS gar nicht mittelfristig geändert werden sollen. Selbst wenn sie langfristig geändert wird, so wird dies auch gleichzeitig mit dem Konsumenten passieren. Eine allgemeine ITS-Architektur auf Basis einer im Web nützlichen Architektur zu reduzieren ist ungünstig. Diese Architektur kann auch selbstständig in einer Implementierung integriert werden, diese ist dann aber beschränkter als die in dieser Arbeit als Ziel gesetzte.

Die Komponentenarchitektur wird am häufigsten in den gefundenen Architekturen verwendet, das liegt auch daran, dass die klassischen Elemente selbst als Komponenten definiert sind. Die vier klassischen Komponenten werden weit verbreitet in ITS eingesetzt und haben sich langfristig als relevant erwiesen. Bei einer eindeutigen Definition ihrer Funktionalität, sollte die Funktionalität die sie bereit stellen auch in einer neuen Architektur

integriert werden. Die meisten Softwarearchitekturen verbessern bei ITS einen Teilaspekt, vernachlässigen dabei aber schon andere Fortschritte in anderen Bereichen. Womöglich fehlt der Überblick über alle Architekturen der hier durch eine Analyse der vorhandenen ITS-Architekturen generiert wurde. Es wäre also sehr günstig auch die Komponentenarchitektur zu verwenden. Es hat sich aber bei der Analyse der vorhandenen Softwarearchitekturen für ITS in dieser Arbeit herausgestellt, dass es bei mehreren Komponenten schnell unübersichtlich wird. Selbst bei Gruppierung in größere Komponenten ist ein häufiges Problem die schwache Einschränkung der Verbindungen.

Die Model-View-Controller-Architektur als eine der am weitesten verwendeten Architekturen weltweit, erscheint passend. Genauere Überlegungen zeigen aber das dies nicht ausreicht, um der Komplexität gerecht zu werden. Die feste Anzahl von genau drei Elementen gibt bei solch einem komplexen System wie einem ITS zu denken, als alleinige Architektur ist sie nicht detailliert genug. Die Verknüpfung vom Model direkt zur View würde den Vorschlag aus dem vorherigen Abschnitt zur Prozesssteuerungskomponente aushebeln. Mit Ähnlichkeiten zur Model-View-Controller-Architektur ist die Schichtenarchitektur. Apple [2018 APPLE INCORPORATED] hat die Model-View-Controller-Architektur als eine 3-Schicht-Architektur neu definiert. Microsoft [2019 MICROSOFT CORPORATION] nennt ihre Neuentwicklung Model-View-Presenter-Architektur. Dabei ist die oberste Schicht für die Benutzeroberfläche zuständig, die mittlere beschreibt den funktionalen Kern mit allen Verarbeitungsmechanismen und die unterste Schicht ist für die Datenhaltung zuständig, also das Speichern und Laden von Daten. Die weiter oben vorgeschlagene Architektur lässt sich problemlos in diese drei Schichten aufteilen, siehe dazu Abbildung R-III1f-1. Doch laut der eigenen vorherigen Definition der Komponenten ist die Datenhaltungsschicht „Model“ und die Benutzeroberflächenschicht „Presenter“ nah an konkreten Implementierung eines ITS und damit ein Bereich in der Entwicklung eines neuen ITS, der nicht durch das Konzept spezifiziert werden soll die Lösung.

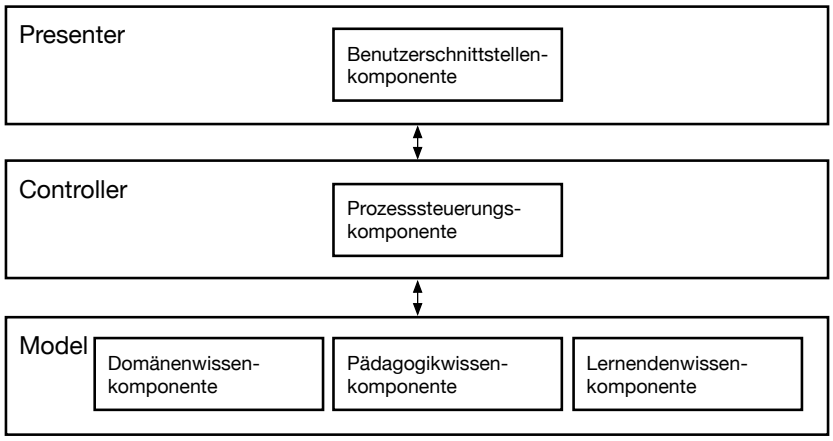


Abbildung R-III1f-1: 3-Schicht-Architektur als Model-View-Presenter

Eine mögliche gute Wahl für den Beginn einer neuen allgemeinen ITS-Architektur ist somit eine Schichten-Architektur verbunden mit der Komponentenarchitektur. In einer neu entwickelten Softwarearchitektur sollten die Vorschläge zur Lösung für die gefundenen Probleme der vorhandenen ITS-Softwarearchitekturen integriert werden.

III.2. ITS-Softwarearchitektur

Zunächst wurden Konzepte für die Architektur entwickelt. Die Aufgabe der Konzepte ist als Grundlage für die Entscheidungen der Architektur zu fungieren. Aufbauend auf diesem Konzept findet die Konkretisierung dieses Konzeptes in Form der eigentlichen Softwarearchitektur statt. Die Konzepte sind die theoretische Grundlage für die zusammenfassenden Entscheidungen, die für die Softwarearchitektur getroffen werden. Die Architektur wird darauf aufbauen auch abstrakt und damit in einem breiteren Rahmen einsetzbar. Auch wenn der Einsatzzweck in dieser Arbeit weiterführend die Architektur und ein Lehrprozess sind, so ermöglichen die vorgestellten Konzepte auch eine Weiterführung weiterer Ideen. Daher ist der theoretischer Rahmen der Arbeit größer ist als die Architektur selbst. Dennoch wird sich in den folgenden Kapiteln nur der Softwarearchitektur zugewandt.

III.2.a. Zielsetzung der Architektur

Da es bisher nur einen begrenzten Aufwand in den letzten Jahrzehnten gab eine neue allgemeine ITS-Softwarearchitektur zu erstellen, aber für eine Vereinheitlichung eine gebraucht wird, ist es das Ziel ein ITS abstrakt zu beschreiben, um eine Hilfestellung an diejenigen zu geben, welche ein ITS implementieren und einsetzen möchten, ohne dabei den Verwendungszweck einzuschränken. Daher ist es wichtig die Kernpunkte eines ITS zu erkennen. Diese Kernpunkte wurden in dieser Arbeit herausgearbeitet und es wird versucht Parameter zu finden, welche ein ITS beschreiben. Die Idee ist es, ein allgemeines Referenzmodell für ITSs zu erzeugen und dabei auf dem vorhandenen Referenzmodell aufzubauen, es zu überarbeiten und die impliziten Anforderungen von heutigen ITSs zu integrieren. Dies soll in einer Art und Weise gemacht werden, dass es auch von Nichtinformatikern verstanden werden kann, denn ein Großteil der Beteiligten an einer ITS-Entwicklung sind aus anderen Domänen und diese Architektur kann als ein Kommunikationsmittel zwischen Ihnen verwendet werden. Die große Anzahl an Beteiligten in anderen Domänen in der ITS-Entwicklung sind häufig schwierig in Konzeptentscheidungen zu integrieren siehe [2010 PAVLIK and TOTH]. Die Lösung ist eine visuelle Repräsentation, da es dann durch die Anschaulichkeit einfacher ist komplexe Zusammenhänge und Strukturen zu erkennen und zu verstehen. Es soll damit schon heute ein Pre-Standard für zukünftige ITSs entstehen. Wir sind in einer Entwicklung der künstlichen Intelligenz zeitlich an einem Punkt der zeigt, dass es möglich sein kann in der mittelfristigen Zukunft solch ein System praktisch umzusetzen. Genau vor der Entwicklung von immer besseren und unter Umständen massentauglichen ITSs muss der Standard existieren, damit er auch vorhanden ist, wenn die nächste Generation von ITSs in der Entwicklungsphase sind. Dieser Pre-Standard sind nicht nur an die Programmierer gerichtet, sondern sollen mehrere Ansichten für alle Beteiligten bieten, darunter fällt auch abstraktere Modelle. Um Missverständnisse zu minimieren, sollte die Spezifikation des Referenzmodells so restriktiv wie möglich sein. Zudem soll die Entwicklung des Konzepts an den ITS-Prozess angelehnt werden, damit wichtige Abläufe beim Lernen direkt im Konzept ersichtlich sind. Zusammenfassend sind die Vorteile folgend.

- Eine detaillierte Spezifikation in einem Referenzmodell
- Beschreibung der Spezifikation durch mehrere formale visuelle Repräsentationen als Kommunikationsmittel
- Mehrere Ebenen der Abstraktion und Details um die Aufgabe für eine Modellbeschreibung zu erfüllen
- Integration eines ITS-Lehrprozesses

Es soll ein Referenzmodell für alle ITSs entstehen, die mehr ist, als die vorhandene klassische ITS-Softwarearchitektur mit nur einer kleinen Änderung. Typische Weiterentwicklungen der ITS-Architektur fügen nur eine neue Funktion hinzu und vergleichen diese mit

alten Architektur. Das Konzept soll für die Forschung auch außerhalb der Informatik relevant sein. Durch Kohärenz der gesammelten Daten und Wiederverwendbarkeit von möglichst vielen der vorhanden ITSs und welche die der Definition nicht ganz folgen. Mit dem Konzept kann man verschiedene ITSs anbieten, welche sich nur geringfügig unterscheiden und damit deren Auswertungen von Lernendenverhalten wissenschaftlich vergleichbar machen. Die Forschung beschäftigt sich mit einzelnen Komponenten und versucht neue Wege für einzelne Komponenten zu gehen, häufig wird in Papieren versucht ein neues Konzept für Lernendenwissenkomponente zu entwickeln, zum Beispiel [2006 WEI and BLANK]. Womöglich weil das Lernendenwissen die Grundlage für die gute Adaptivität des ITS ist. Doch wenn der Rest des ITS auch neu entwickelt wird und sich damit unterscheidet, dann ist ein Vergleich zu dem vorherigen ITS schwer möglich und damit die Bewertung der Gewinne für die neue Komponente zu ungenau.

III.2.b. Anforderungen an die Softwarearchitektur

Es wurde gezeigt, dass die klassische Architektur viele Probleme hat. Doch auch die Veränderung der klassischen Architektur war aufgrund fehlender Definitionen der ITS-Funktionalität unzureichend. Die neue Architektur nimmt die Empfehlungen aus dem Kapitel „Wahl der Softwaregrundlage“ für eine Gliederung des Systems in Schichten und seine Komponenten. Für die Entwicklung von einer Schichtenarchitektur wird untersucht, welche bekannten Komponenten in welche neuen Schichten eingeteilt werden können. Die klar definierten Komponenten aus der veränderten klassischen Architektur und der erste Ansatz drei Schichten zu verwenden, führt zu der Benennung und Definition der Schichten: Eine für die Datenerhaltung, eine für die Benutzeroberfläche/Lernendeninteraktionen und die letzte beinhaltet die Prozesssteuerung und die Funktionalität. Im nächsten Schritt werden die Schichten mit der schon in dieser Arbeit veränderten klassischen Architektur zusammengeführt siehe Abbildung R-III2b-1.

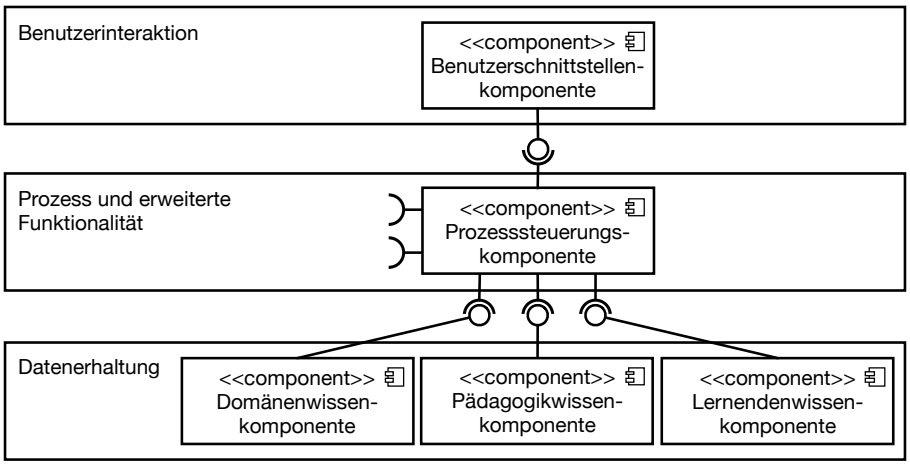


Abbildung R-III2b-1: Kombination der überarbeiteten klassischen Architektur mit MVP

Dabei fällt auf, dass die Datenerhaltung und die Benutzeroberfläche die jeweiligen Enden der Interaktionskette sind. Für die meisten der ITS-Entwicklungen ist die Datenerhaltung das wichtigste, deshalb hat sie dementsprechend auch die Mehrheit der Komponenten. Es wird nicht in Funktionen gedacht und was das ITS können muss, sondern welche Daten konkret gesammelt werden und dass die Domänenexperten als Autoren selbst direkt an den Datenbanken arbeiten könnten. Die Datenbanken (welche auf Performanz und nicht Lesbarkeit optimiert wurden) erfordern spezialisiertes Datenbankwissen, sind bei wachsenden Daten unübersichtlich, kaum verständlich bei erhöhter Komplexität und müssen für eine Wiederverwendbarkeit konkret in einer Implementierung bleiben. Datenbanken für ein ITS zu erstellen ist aufwändig und bereits erstellte Datenbanken sind sehr

wertvoll. Da eines der Hauptziele die Wiederverwendbarkeit der Daten ist, zum Beispiel für eine weitere Auswertung und Vergleiche zwischen ITSs, muss es leicht sein sie in andere Systeme zu integrieren. Hierbei sei angemerkt, dass die Datenmenge und -komplexität auch durch das beibehalten der Rohdaten der Lernendenaktionen mit der Zeit zunimmt. Rohdaten können immer neu ausgewertet werden und sollten daher immer aufgehoben werden. All diese Anforderungen an die Datenerhalten geht nur mit einem konkreten definierten Format. Solch ein Format würde viele implementationsspezifische Details in die Komponenten integrierten und zusätzlich die Komplexität merklich erhöhen. Deshalb wird in dieser Arbeit davon ausgegangen dass ein Domänenexperte als Autor auch die Benutzeroberfläche verwenden sollte und die Datenerhaltungsschicht zu viel Wissen über Datenbanken benötigt und zu einzigartig für jede Implementation ist, um sie in dem zu entwerfenden allgemeinen Softwarearchitektur beschreiben zu können.

Nun hat die Benutzeroberfläche ein ähnliches Problem. Als Schnittstelle für den Nutzer des ITS ist diese Komponente sehr wichtig. Die Auswirkungen auf den Menschen durch eine Änderung in der Schnittstellen ist vergleichsweise hoch. Kleine Änderungen an der Positionierung oder der Farbe von Elementen in der Benutzeroberfläche können Nutzer sehr schnell ärgern, besonders wenn diese aus ihrer Sicht nicht nachvollziehbar und damit unnötig erscheinen. Zudem tritt auch ein Gewöhnungseffekt ein, bei dem sich Nutzer nach längerer Benutzung mit der bekannten Benutzeroberfläche „Zuhause“ fühlen, egal ob diese optimal ist. Änderungen sind damit auch hier kritisch. Eine Übertragbarkeit und langfristige Nutzung führt schnell zu einer Konkretisierung der Benutzeroberfläche über einzelne ITS-Versionen und -Varianten hinweg. Auf die Benutzeroberfläche kann demnach auch nicht genau eingegangen werden, ohne den Abstraktionsgrad zu verringern. Dass sie in einen Bereich gehört dessen Ergründung tief in den Bereich von Design und Psychologie geht, welches keines der Kernpunkte dieser Arbeit ist, ist ein zusätzlicher Punkt. Für Beispielfälle oder Prototypen kann die Benutzeroberfläche aber in dieser Arbeit konkret realisiert werden. Diese Arbeit geht neben der Benutzeroberflächengestaltung auch auf keine anderen psychologischem Effekte tiefer ein, wie die konkrete Anpassung der Benutzeroberfläche an die Zielgruppe oder den einzelnen Lernenden, als auch, wann didaktisch das System eingesetzt werden sollte und was erwartet wird. Es wird in der Neuentwicklung der Architektur zum Vorteil der Wiederverwendung eine klare Grenze gezogen: Interaktionen mit konkreten Daten und konkrete Benutzeroberflächen/Benutzerinteraktionen gehören nicht in die Architektur. Die Arbeit konzentriert sich demnach auf die genauere Definition der Funktionen von ITS, welche sich in der mittleren Schicht befinden, siehe dazu Abbildung R-III2b-2, wird aber auch die anderen Komponenten etwas näher betrachten und ergänzen wo es relevant für hinzukommende Elemente ist.

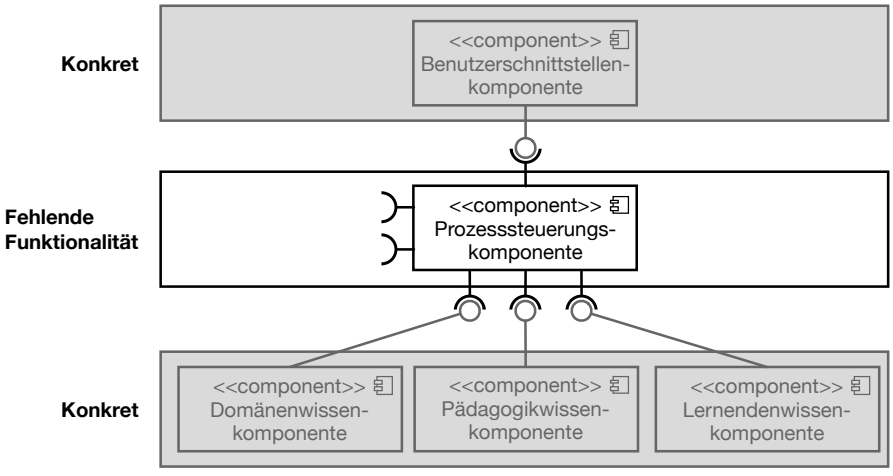


Abbildung R-III2b-2: Wichtige Schicht ist die mittlere mit der fehlenden Funktionalität

Zu den aktuellen Komponenten (Benutzerschnittstellenkomponente, Prozesssteuerungskomponente, Domänenwissenkomponente, Pädagogikwissenkomponente, Lernendenwissenkomponente) kommen noch die fehlenden Funktionalitäten (die auch in den klassischen Komponenten untergebracht) hinzu. Die fehlen Funktionalitäten können offensichtlich durch ihre Funktionalitäten definiert und neuen Komponenten zugeordnet werden. Durch die „Funktionalen Komponenten“ die über die Prozesssteuerungskomponente kommunizieren, ist es möglichst alle ITS-Typen (Constraint based tutors, Model tracing tutors, Example tracing tutors) darzustellen. Die Fragen, die man sich dafür stellen muss sind folgend aufgezählt.

- Was muss das ITS können?
- Wie lässt sich das aufteilen ?
- Gibt es dafür einen Anwendungsfall der diese Funktionsweise benötigt?

Die Funktionalitäten werden nach Ähnlichkeit zusammengefasst. Zudem wird überlegt, an welcher Stelle im ITS-Lehrprozess man sie benötigt. Ein Beispiel ist die Generierung des Profils für den Lerner: dies ist zu Beginn der Sitzung erforderlich - das generierte Profil wird jedoch im Verlauf der Lehrsitzung nicht nochmal neu generiert. Deshalb kann diese Funktionalität nicht in einer Komponente sein die auch Aufgaben für die Lehrsitzung erledigt. Die Verbindungen müssen dabei, zur Vereinfachung des Gesamtmodells, restriktiv gehalten werden. Es wird darauf geachtet die Flexibilität nur dann zu gewähren, wenn es sich als notwendig erweist.

Das ITS-Konzept trennt klar Pädagogikwissen von Domänenwissen. Die domänenbasierte Softwarearchitektur versucht eine Struktur abzubilden, welche in der Domäne des Domänenwissens als de facto Standard angesehen wird. Dies hat den Vorteil, dass die Verwendung des Systems für die Autoren des Lehrmaterials eine flachere Lernkurve benötigt, schränkt aber die Verwendung in anderen Domänen stark ein oder macht es ineffizient. Das im Rahmen der vorliegenden Arbeit entwickelte Konzept soll für alle Domänen gelten und muss daher ungebunden von der Domäne sein. Nachträglich kann die entstandene Architektur immer an die jeweilige Domäne angepasst werden, in den konkreten Schichten ist das sogar gezielt erwünscht.

Das ITS-Konzept wird über verschiedene Ebenen der Systemmodellierung definiert. Es wird in der Modellierung des Systems in mehreren abstrakten Ebenen gearbeitet die folgend aufgezählt sind.

- Systemmodellierung mit Software-Engineering-Ebenen als eine Hilfestellung für Softwareentwickler in der Implementierung eines ITS
- Verhaltensmodellierung der Komponenten, insbesondere die Adaptivität zum Lernenden, als eine Hilfestellung für KI und Didaktiker zur Anpassung des Lehrmaterials und des Lehrverhalten des ITS
- Als Mediator zwischen den Domänenexperten und Systemingenieuren als eine Hilfestellung für Domänenexperten und als Kommunikationsgrundlage.

III.2.c. ITS-Schichten neu definiert

Es wird trivial in der höchsten Abstraktion (Abstraktionsebene 1) für das ITS als Lehrsoftware angefangen, in der es nur einen Nutzer pro ITS gibt der mit dem System interagiert, siehe Abbildung R-III2c-1. Nun kann das Artefakt ITS näher betrachtet und detaillierter aufgezeigt werden.

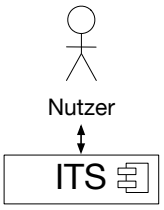


Abbildung R-III2c-1: Abstraktionsebene 1 der Softwarearchitektur

Um die Architektur des ITS besser zu strukturieren wird das System in Schichten geteilt. Dies hat den Vorteil, dass es klarer strukturiert ist und das häufige Problem vermieden wird, dass zu viele unnötige Verbindungen direkt zwischen Komponenten entstehen. Eine Schicht kann nur maximal zwei Kommunikationskanäle besitzen, einen zur nächsthöheren Schicht und einen für die nächst niedrigere Schicht. Das Modell ist für eine grobe Einteilung durch seine Einfachheit und Modularisierung sehr gut geeignet für ein ITS welches sowohl Anwendungslogik, Datenverwaltung als auch die Oberflächenverwaltung vereint. Es gibt viele unterschiedliche Muster für das Schichtmodell, die sich in erster Linie durch die Anzahl der Schichten kategorisieren lassen. Wichtig ist auch der Unterschied von einer „ Schichtenarchitektur “ zu „ Multitier-Architektur“, ein „tier“ ist hierbei eine physikalisch trennbare Schicht. Im deutschen Sprachgebrauch werden beide Schichtenarchitektur oder Schichtenmodell genannt. „Layer“ stellen die softwareseitigen Schichten dar, welche bei modularer Programmierung zum Einsatz kommen können, während „tier“ die hardwareseitige Unterteilung modelliert, was bei verteilten System zum Einsatz kommen kann. Bekannte Beispiele der Multitier-Architektur ist die Client-Server-Architektur für Multitier-Architektur mit zwei tiers, das „ISO/OSI-7-layer-model“ für eine Multitier-Architektur mit sieben tiers. Für Schichtenarchitektur sind bekannte Beispiele das „Model-View-Presenter“ als Schichtenarchitektur mit drei Schichten und die „4-layer-ring-Architektur“ für die Betriebssystemprogrammierung. Es gibt eine „4-Layer-Agent-Architektur“ für das E-Learning bei dem die Datenerhaltungsschicht eine abstrahierte Zwischenschicht erhält [2010 ALI, DEHGHAN and GHOLAMPOUR].

Wie in den Vorschlägen für die klassische Softwarearchitektur aufgezeigt, lässt sich die bisherige Komponentenarchitektur leicht in drei Schichten aufteilen. Bei der bisherigen Aufteilung in drei Schichten ist dies vergleichbar mit der Model-View-Presenter-Architektur. Ein Muster welches für ITS als wichtig eingestuft wurde [2006 HARRER and MARTENS]. Das führt zu dem Problem, dass die beiden äußeren Schichten zu konkret werden müssen um sie genügend nützlich zu beschreiben. Deshalb werden zwei weitere Schichten hinzugefügt, welche den Abstraktionsgrad erhöhen und als Zwischenschichten zur ITS-Anwendungslogik dienen. Fünf Schichten sind häufig bei Schichtarchitekturen anzutreffen und bilden häufig die folgenden Aufgabenbereiche in genau dieser Reihenfolge ab.

- Presentation (Präsentation)
- Control oder Application (Steuerung)
- Business (logic), Service, oder Domain (Anwendung)
- Persistence (Datenzugriff)
- Database (Datenhaltung)

Die gezeigte Aufteilung und Benennung ist am weitesten verbreitet. Von dieser Architektur gibt es auch Abwandlungen, zum Beispiel die Aufteilung von dem „Business layer“ in „Service layer“ und „Business logic layer“, das Weglassen des „Database layers“, die Integration des „Control layers“ in den „Presentation layer“ oder der „Persistence layer“ und „Database layer“ werden zu „Infrastructure layer“ zusammengefasst. Auch die Namen werden teilweise anders verwendet: Der „Presentation layer“ heißt dann der „UI Components layer“ und der „Control layer“ wird zum „Presentation layer“. Angelehnt an diese Architektur wird auch die neue Architektur spezielle für ITSs fünf Schichten besitzen. Die fünf Schichten in dem Konzept sind folgend in Reihenfolge aufgezählt:

- Interaktionsschicht
- Temporärschicht
- Funktionalschicht
- Persistierungsschicht
- Datenhaltungsschicht

Mit diesen fünf Schichten ist es möglich den implementationsspezifischen Beschreibungen eines ITS einen Platz zu geben, ohne dass diese in die Softwarearchitektur miteinbezogen werden, damit sie in einer abstrahierten Architektur einen klar definierten Bereich haben. Dieser Platz sind die beiden äußeren Schichten Interaktionsschicht und Datenerhaltungsschicht, siehe dazu Abbildung R-III2c-2. Die Schichten werden folgend einzeln erklärt.

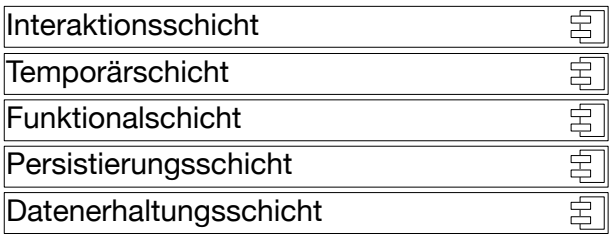


Abbildung R-III2c-2: Schichten der Softwarearchitektur (Abstraktionsebene 2)

Interaktionsschicht

Die Interaktionsschicht beschreibt die konkrete Darstellung der Elemente des Kommunikationsmediums. Das Medium beinhaltet fast immer die grafische Benutzeroberfläche und diese stellt die hauptsächliche Kommunikation mit dem Lernenden dar. Die aktuell sichtbare Benutzeroberfläche wird in dieser Schicht konkret definiert mit ihrem Layout und den darin befindlichen Elementen mit ihren Eigenschaften. Die Eigenschaften der Elemente welche besonders wichtig sind, sind die Position des Elements auf der anzuzeigenden Fläche, deren Abmessungen und die Art wie es dargestellt werden soll. Es bildet damit die konkrete Darstellung einer Benutzeroberfläche zu jeweils einem Zustand aus Benutzersicht. Weitere Inhalte der Schicht sind aktuelle Audiowiedergaben und wie diese wiedergegeben werden. Die Benutzeroberfläche benötigt auch eine gewisse Prozesskontrolle, welche sich nur um die Oberflächeninteraktionen kümmert, den Zustand hält, welcher nur für die aktuelle Sitzung relevant ist, und mehrere Elemente die nicht alle gleichzeitig angezeigt werden zusammenfasst. Es gibt nur bestimmte Pfade um zwischen den einzelnen Ansichten zu navigieren, dessen Regeln selbst nicht in den einzelnen Ansichten gespeichert werden sollen — dass übernimmt dann die Temporärschicht.

Temporärschicht

Die Temporärschicht dient als Kurzzeitgedächtnis und ist eine Zwischenschicht um Ausgaben vom ITS an die Benutzerschnittstelle gerecht anzupassen und umgekehrt Eingaben vom Lernenden für die oberflächenunabhängigen Programmteile korrekt umzuwandeln. Die Schicht stellt Daten bereit, die zwar im Augenblick nicht sichtbar sind, aber schon geladen wurden, weil sie entweder mit hoher Wahrscheinlichkeit benötigt werden, sie noch nicht vollständig sind oder weil es sich nicht lohnt sie persistent zu speichern und sie noch immer verwendet werden könnten und deshalb noch nicht gelöscht werden können. Darunter fallen unter anderem die Daten von Ansichten in denen man sich schon befand und zu denen man schnell zurückkommt mit jeweiligen benötigten Werten, wie zum Beispiel, wie weit man herunter gescrollt war und was in den einzelnen Feldern für Zwischenberechnungen schon fertig waren. Diese Schicht bildet die bisher abgelaufenen als auch wahrscheinlich nächsten Schritte als zwischengeladene Daten von Ansichten durch das Programm.

Funktionalschicht

Die Funktionalschicht stellt das Verhalten und die Funktionalitäten des ITS dar. Der integrierten Lehrprozess und dessen Ablauf sicherzustellen ist eine zentrale Funktionalität. Viele Funktionalitäten die vorher in anderen Komponenten untergebracht wurden, dürfen durch die klare neue Definition nur noch in diese Schicht. Ohne funktionelle Komponenten kann das ITS seine Aufgabe nicht erfüllen, die Aufgaben für Analyse, Generierung, Bewertung und so weiter sind nicht Aufgabe der anderen Schichten. Nun lassen sich die noch möglichen funktionalen Komponenten leicht in zwei Kategorien teilen. Ein Teil stellt die ITS-Kernfunktionalität dar und der andere die Programmfunktionalitäten. Die Programmfunktionalität erwartet man auch von vielen anderen Systemen und sind daher eher allgemeiner Natur.

Persistierungsschicht

Die Persistierungsschicht beschreibt die einzelnen Bereiche welche Zugriff auf bestimmte vorverarbeitete Daten haben. Diese Schnittstelle besteht damit aus den abstrahierten Daten hin zu den Informationen welche in den drei datenbezogen klassischen Komponenten zu finden sind: Domänenwissenkomponente, Lernendenwissenkomponente und Pädagogikwissenkomponente. Diese Komponenten sorgen für die Validierung und Konvertierung von Einträgen in den jeweiligen Datenbanken zu Objekten. Datenbank-Objekte bieten vorgefertigte Methoden und daraus berechnete Attribute an und erlauben damit eine vereinfachte Benutzung in der Programmierung durch Auslassen der Komplexität der konkreten Datenbankstruktur.

Datenhaltungsschicht

Die Datenhaltungsschicht stellt die Schicht für die konkreten Datenbanken und deren Dateien für deren ordnungsgemäßen Betrieb dar. Das Dateiformat, deren Formatierung, der Dateipfad und unter Umständen ganz andere Angaben sind implementationsspezifische Details, welche für die Weiterverwendung von Datenbanken dennoch sehr wichtig ist. Beispiele für Dateiformate der Datenbanken selbst sind JSON, XML, SQLITE oder einfache Textdateien. Unter Umständen kann es zur Verwaltung notwendig sein, Skripte oder Klassen für die allgemeine Verwaltung dieser Dateien zu erstellen um sie performant zu halten, zum Beispiel die Organisation, Benennung, Defragmentierung oder erste Initialisierung der Dateien.

Die obersten zwei Abstraktionsebenen

Werden die beiden obersten Abstraktionsebenen kombiniert, so erhält man voneinander abhängige Schichten innerhalb des Systems, siehe Abbildung R-III2c-3. Die Verknüpfungen zwischen den Komponenten ist bei einer Schichtenarchitektur offensichtlich.

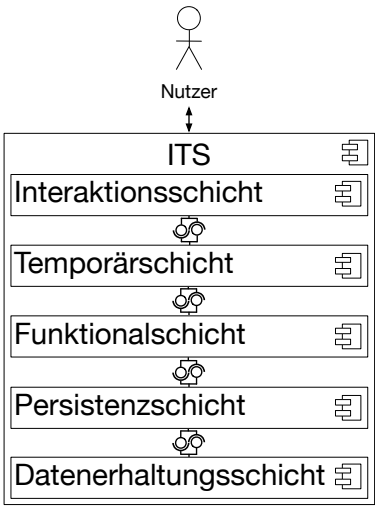


Abbildung R-III2c-3: Die obersten zwei Abstraktionsebenen

III.2.d. Die dritte Abstraktionsebene

Es liegt jetzt ein neuer Ansatz für die Softwarearchitektur vor: Fünf Schichten zu verwenden und diese immer weiter in einzelnen Komponenten zu konkretisieren. Dieser wird verwendet um die jetzt vorhandenen Schichten mit Komponenten zu füllen. Dabei betrachten wir zunächst die klassischen Komponenten und wie sie in unser neues Modell passt.

Klassische Komponenten in dem 5-Schicht-Modell

Die klassische Softwarearchitektur ist absichtlich in ihren Komponenten noch immer vorhanden, siehe Abbildung R-III2d-1. Es gibt in der klassischen Architektur die Komponente der Benutzeroberfläche, welche in dem Schichtmodell die Interaktionsschicht und die Temporärschicht darstellt. Die klassischen Komponenten des Lernermodells, Domänenwissen und des Pädagogikwissens spiegeln sich in der Persistenzschicht und Datenerhaltungsschicht wider. Dabei wurden Pakete für den Zugriff durch die Persistierungsschicht auf genau die Bereiche der gespeicherten Daten in der Datenhaltungsschicht geschaffen, welche inhaltlich und namentlich ungefähr zu den vielen klassischen Definitionen passen. Es fällt dabei auf, dass die im klassischen Modell aufgezählten Komponenten dann nicht in der Funktionalschicht beinhaltet sind. In ihrer Mehrheit kommen die Komponenten in der Persistierungsschicht vor. Auch aufgrund dessen, dass die Prozesssteuerung als Komponente seltener in beschriebenen Architekturen vorkommt. So zeigt sich, dass die bisherige klassische Architektur sich nicht auf die abstrakte Beschreibung der Implementierung fokussiert hat, sondern ist von den beiden konkreten Seiten des Schichtmodells auf das Problem zugegangen: Wie man die Benutzeroberfläche gestaltet und wie man die Speicherung der Daten gestaltet. Der Kern eines ITS lässt sich an der Definition ableiten und besteht nicht aus Daten und Ansichten, sondern wird durch sein adaptives Verhalten definiert. Deshalb ist eine neue Architektur basierend auf den Funktionalitäten besser für eine Neuimplementierung geeignet, weil sie dem Softwareentwickler leitet, wie ein Softwaresystem nach den Anforderungen der komplexen und flexiblen Bereiche der Umsetzung aufgebaut werden sollte und nicht nur die Sichten abdeckt welche ein Autor und Lernender hätte. Zudem wird wie vorher schon erwähnt die Vergleichbarkeit gesteigert und es gibt eine vollständigere Beschreibung eines ITS im Allgemeinen. Darauf aufbauend werden die Komponenten für die Schichten entwickelt.

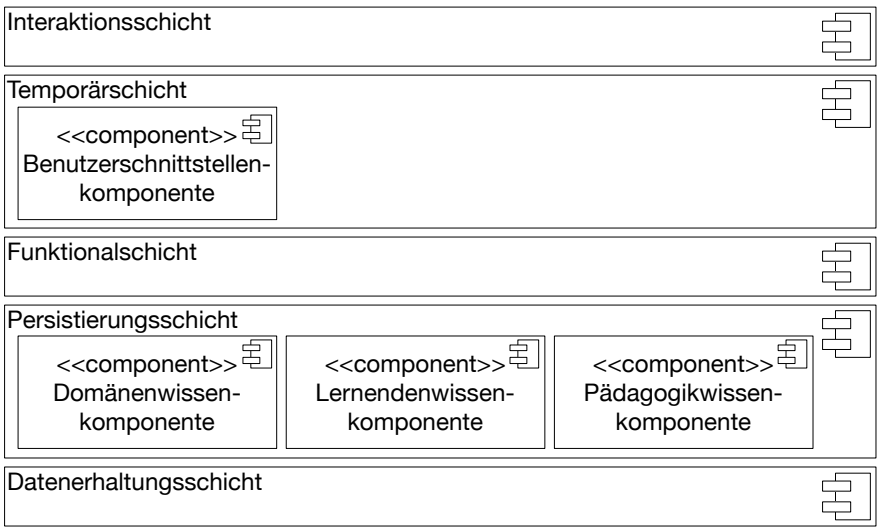


Abbildung R-III2d-1: Klassische Komponenten in dem neuen Schichtmodell

Verwendete Komponenten in der dritten Abstraktionsebene

Die Interaktionsschicht und die Datenerhaltungsschicht bleiben bewusst leer, weil wie in vorherigen Abschnitten beschrieben diese zu konkret für eine allgemeine Softwarearchitektur werden.

Eine Komponente für die Temporärschicht wie in dem klassischen Modell ist zu ungenau. Es wird daher die Benutzerschnittstellenkomponente in zwei Komponenten aufgeteilt. Die Ansprüche eines Programms an seine Benutzeroberfläche für die Lehre und die nicht für die Lehre sind unterschiedlich. Deshalb findet hier eine Aufteilung statt. Die Benutzeroberfläche und deren Interaktion die nicht für die Lehraufgaben ausgerichtet ist, kann sehr klassisch gestaltet sein, wenig Freiheiten bieten und nicht adaptiv gestaltet werden. Bei den Lehraufgaben hingegen ist der Anspruch ein ganz anderer, die Funktion muss nicht fehlerlos sein und ohne Erklärungen funktionieren, sondern orientiert sich am Lernenden, solange ein Dialog entsteht, der Lernende sich begeistert und lernt, wird das System ihn langsam zum Ziel führt. Ein Hauptmenü braucht keinen ausführlichen Dialog und sollte auch keine Einführung benötigen.

Für die Funktionalschicht wird auch mindestens eine Komponente benötigt. Die Aufspaltung der Benutzerschnittstellenkomponente in der Temporärschicht ist auch sinnvoll für die Funktionalitäten der Funktionalschicht. So gibt es Funktionen welche für die Lehre und damit zur Ausführung der Lehrtätigkeit als Grundaufgabe des ITS beitragen und andere Funktionen die von guten Lehrprogrammen zusätzlich erwartet werden kann. Auch deren Anforderungen und Ziele unterscheiden sich und können gut getrennt werden.

Zu guter Letzt kann die Aufteilung der Persistierungsschicht in die Aufteilung der klassischen Komponente als fast ausreichend angesehen werden. Es ist bei eigenen Entwicklungen und vorhandenen ITSs immer wieder aufgefallen, dass es nötig ist Daten zu speichern, die nicht einem der drei Wissensarten (Domänenwissen, Lernendenwissen oder Pädagogikwissen) zugeordnet werden können. Darunter fallen die Einstellungen des jeweiligen Programms über die Lehrtätigkeit hinaus, dies reicht von Benutzernamen, Designwahl, letzter Start, Hardwareinformationen bis hin zu Lizenzschüsseln. Meist sind es bunt zusammengewürfelte Daten die wenig Speicherplatz verbrauchen und keine Daten über Zeit akkumulieren. Diese Komponente wird hinzugefügt.

So entstehen Komponenten, welche die Schichten füllen, siehe dazu Abbildung R-III2d-2. Die jeweiligen Komponenten sind folgend generell beschrieben und werden in den nächsten Abschnitten detailliert beschrieben.

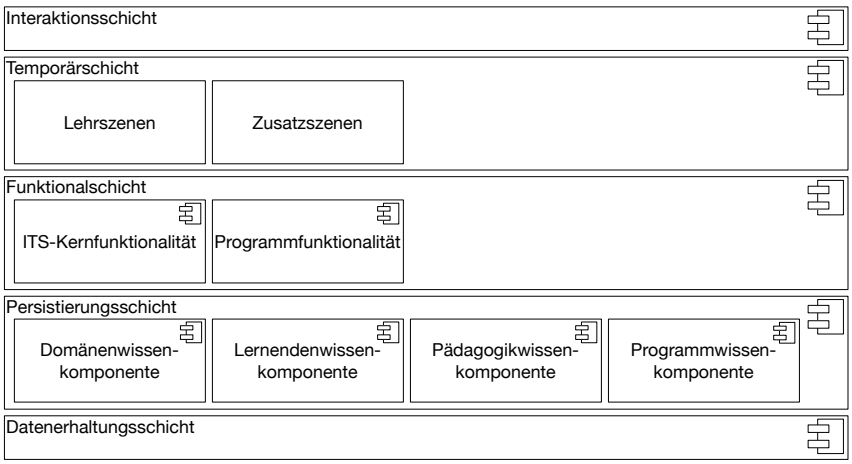


Abbildung R-III2d-2: Komponenten der zweiten und dritten Abstraktionsebene

Die Komponenten der dritten Abstraktionsebene

- Lehrszenen: Die Aufgabe der Lehrszenen ist es alle Lehrszenen zu verwalten und deren wiederkehrenden Aufgaben in einer höheren Instanz zu sammeln. Sie implementieren das Verhalten von konkreten Ansichten aus der Interaktionsschicht.
- Zusatzszenen: Die Zusatzszenen beinhalten alle Szenen die nichts mit dem Lehrprozess zu tun haben. Darunter fallen Szenen zur Verwaltung des Programms oder zur Navigation und Auswahl wie zum Beispiel das Hauptmenü. Sie implementieren das Verhalten von konkreten Ansichten aus der Interaktionsschicht.
- ITS-Kernfunktionalität: Die ITS-Kernfunktionalität beinhaltet all die Funktionalität, welche für den Lehrprozess relevant sind. Es wird nur Funktionalität implementiert die ungeachtet von Datenspeicherung und Benutzeroberfläche implementiert werden kann.
- Programmfunktionalität: Die Programmfunktionalität besteht aus all den Funktionalitäten welche nicht zu der ITS-Kernfunktionalität gehört. Es wird nur Funktionalität implementiert die ungeachtet von Datenspeicherung und Benutzeroberfläche implementiert werden kann.
- Domänenwissenkomponente: Die Domänenwissenkomponente bündelt alle Komponenten welche sich um die Umwandlung, Laden, Fehlerbehebung und Speicherung der domänenspezifischen Daten kümmern
- Pädagogikwissenkomponente: Die Pädagogikwissenkomponente bündelt den Zugriff, Veränderung, Löschung und Datentransformation auf das gesamte Pädagogikwissen in einer für die Funktionalschicht einfach zu verarbeitenden Form.
- Lernendenwissenkomponente: Die Lernendenwissenkomponente bündelt den Zugriff, Veränderung, Löschung und Datentransformation auf das gesamte Lernendenwissen in einer für die Funktionalschicht einfach zu verarbeitenden Form.
- Programmwissenkomponente: Die Programmwissenkomponente bündelt den Zugriff, Veränderung, Löschung und Datentransformation auf das gesamte Programmwissen in einer für die Funktionalschicht einfach zu verarbeitenden Form. Dabei gibt es keine Interpretation des Wissens.

Obersten drei Abstraktionsebenen kombiniert

Nun werden diese ersten drei Abstraktionsebenen kombiniert und die Abhängigkeiten der dritten Abstraktionsebene hinzugefügt. Dies führt zu einer Softwarearchitektur, die mehr Details bereitstellt, siehe Abbildung R-III2d-3. Die Komponenten der Temporärschicht greifen dabei auf konkrete Komponenten in der Interaktionsschicht zu, wie zum Beispiel Ansammlungen von Ansichten oder Schablone für eben jene. Zudem brauchen die Komponenten der Temporärschicht beide Zugriff auf die beiden Komponenten in der Funktionalschicht. Die Komponenten der Funktionalschicht greifen auf die Komponenten

der Datenerhaltungsschicht zu, mit der Ausnahme, dass die ITS-Kernfunktionalität keinen Zugriff auf das Programmwissen benötigt, weil dieses nichts für den Lehrprozess beinhaltet. Die Persistenzschicht hat auch in der dritten Abstraktionsebene keine Definition zu der Datenerhaltungsschicht weil eben jene zu konkret ist.

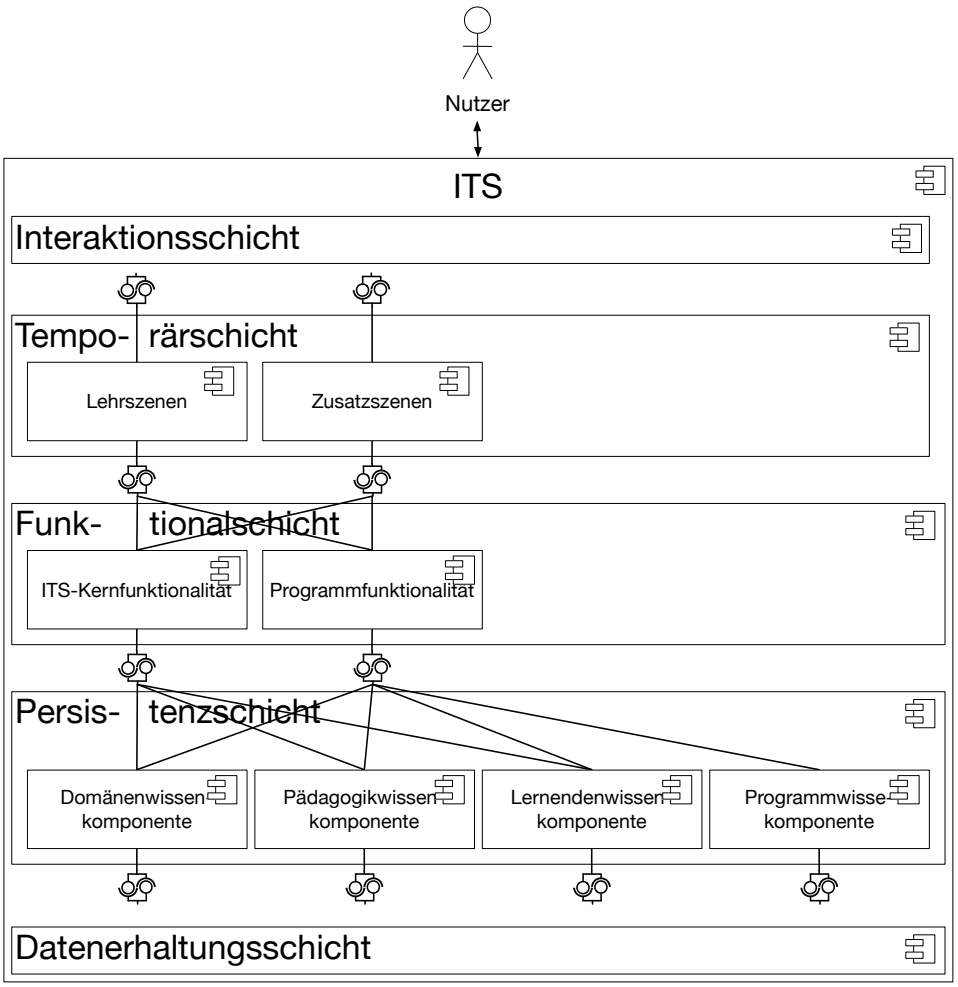


Abbildung R-III2d-3: Obersten drei Abstraktionsebenen

III.2.e. Komponenten der Temporärschicht im Detail

Neben der Aufteilung der klassischen Komponenten in der Persistenzschicht ist solch eine Aufteilung auch für die Temporärschicht vorteilhaft. Die Interaktionsschicht ist zu konkret in ihren Teilen, als dass sie domänenübergreifend spezifiziert werden kann, bei einer Benutzeroberfläche wären dies die implementierten Ansichten. Deshalb wird die abstrahierte Temporärschicht für die Einteilung der Verwaltung für die Ansichten genommen. Die in der Temporärschicht enthaltene klassische Benutzerschnittstellenkomponente ist in den Softwarearchitekturen bisher vernachlässigt worden. Es ist nur eine Komponente die allgemein alles umfasst was mit der Schnittstelle zum Benutzer einhergeht. Darunter fällt in erster Linie die grafische Benutzeroberfläche mit Maus und Tastatureingaben und Audioausgaben. Sekundär gibt es weit weniger häufig verwendete Ein- und Ausgabegeräte, wie zum Beispiel Mikrofone, Kameras, Beschleunigungssensoren, Vibrationsgeräte und anderweitige Controller. Die Aufgabe der Temporärschicht ist es die nur zur aktuellen Situation benötigten Informationen zu beschaffen und damit automatisch die Benutzung des ITS in Gruppen einzuteilen, welche von der aktuellen Benutzerschnittstelle abhängig ist. Es gruppiert die Anforderungen an das System nur an das für den Nutzer ersichtlichem, Funktionalitäten die für das System mehrere Aufgaben darstellt für den Nutzer aber nur als eine ersichtlich ist, sind auch nur eine Szene. Die klassische Benutzerschnittstel-

lenkomponente kann damit in Szenen eingeteilt werden, die jeweils die unterschiedlichen Bereiche zu denen man navigieren kann, abdecken. Die Aufgabe einer Szene besteht dabei ein oder mehrere konkrete Ansichten zu referenzieren. Da diese aber nur als Schablone dienen, muss die Szene jeweils die passendste Ansicht wählen und mit die für die Ansicht notwendigen Daten füttern. Zudem ist es Aufgabe der Szene zu entscheiden, was mit der jeweiligen Eingaben passiert. Da die Szenen größer sind als eine einzelne Ansicht, können einzelne Aufgaben an jeweilige Controller innerhalb der Temporärschicht delegiert werden. Die Szenen lassen sich grob in zwei Bereiche aufteilen, einmal die für den Lehrprozess relevanten Szenen und die übrigen, siehe hierzu Abbildung R-III2e-1. Es wird nicht auf die im Hintergrund laufenden und benötigten Funktionen Rücksicht genommen, sondern es werden nur die für den Nutzer, sei es Lernende, Betreuer, Autoren oder Lehrer, erkennbaren Bereiche analysiert. Aus diesem Grund ist immer nur eine Szene aktiv.

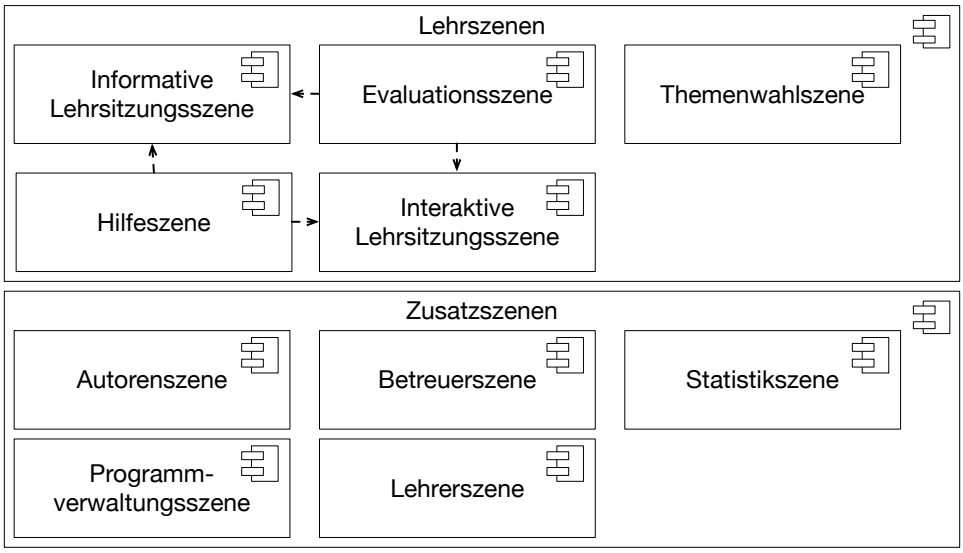


Abbildung R-III2e-1: Szenen in der Temporärschicht mit Abhängigkeiten

Lehrszenen

Die Szenen welche für den Lehrprozess eine Rolle spielen sind unter dem Namen „Lehrszenen“ zusammengefasst. Darunter fällen die Interaktionsbereiche welche für den Lernenden erkennbar sein sollen und sein Handeln erfordern. Aus dem Lehrprozess zu erkennen sind dabei die Wahl für das Thema der kommenden Lehrsitzung, die Lehrsitzung selbst und deren Evaluation. Die Themenwahl wird „Themenwahlszene“ und die Lehrsitzung wird aufgeteilt in „Informative Lehrsitzungsszene“ und „Interaktive Lehrsitzungsszene“. Die Themenwahlszene stößt im Hintergrund einige Funktionen zur Generierung und Anpassung des Lehrmaterials an, die dem Nutzer womöglich gar nicht auffallen. Die „Hilfeszene“ ist abhängig davon, dass überhaupt eine „Lehrsitzungsszene“ vorhanden ist, weil sie sonst nicht beim Lernen helfen kann. Sie kann in beiden Lehrsitzungsszenen verwendet werden und ist damit übergeordneter als andere Bereiche. Als Letztes ist die „Evaluationsszene“ für die Evaluation zuständig und nur dann einsetzbar wenn es eine Lehrsitzung gab. Die Lehrsitzung des informativen und des interaktiven Lehrmaterials wird sich relativ stark unterscheiden, deshalb werden dadurch zwei Szenen definiert. Das ITS muss die Benutzerführung bei einem größeren Hilfeanzeige auch die Szene ändern, dies erzeugt eine neue Szene.

Lehrszenen beinhaltet

- Themenwahlszene: Stellt die Verwaltung und Interaktionsmöglichkeiten für die Ansichten bereit, welche dem Lernenden ermöglichen ein Thema zu wählen. Darunter fallen Vorschläge anzeigen, Themenliste in verschiedenen Detailgeraden anzeigen, filtern oder sortieren, eine Auswahl von mehreren Themen erlauben, markieren von Themen, hervorheben von besonders beachtenswerten Themen und so weiter. Diese Szene ist beendet wenn der Lernende ein Thema gewählt hat. Dies ist die Komponente welche den ersten Teil des ITS-Lehrprozesses erfüllt.
- Informative Lehrsitzungsszene: Stellt die Verwaltung und Interaktionsmöglichkeiten für die Ansichten bereit, welche dem Lernenden informatives Lehrmaterial zu lernen. Dies inkludiert nicht die Wahl der Anpassung des Lehrmaterials an den jeweiligen Lernenden. Es entscheidet jedoch wie am besten das Lehrmaterial in das vorgegebene Layout der Interaktionsschicht passt und wie auf Aktionen des Lernenden reagiert wird. Es kann durchaus mehr als eine Ansicht benötigt werden.
- Interaktive Lehrsitzungsszene: Stellt die Verwaltung und Interaktionsmöglichkeiten für die Ansichten bereit, welche dem Lernenden interaktives Lehrmaterial zu lernen. Dies inkludiert nicht die Anpassung des Lehrmaterials an den jeweiligen Lernenden. Es wird ein Dialog zwischen dem Lernenden und dem ITS möglich sein, bei dem die Szene entscheidet was nach einer Eingabe passiert. Dieser Grad an Freiheit fordert auch von der Szene mehr ab, diese muss daher häufiger auf die vielen Funktionalitäten der Funktionalschicht zugreifen, wenn die Aufgaben komplizierter werden.
- Hilfeszene: Die Verwaltung der Ansichten und deren Interaktionen wenn die Hilfsfunktion gebraucht wird ist Aufgabe dieser Szene. Sie entscheidet nicht wann die Hilfe benötigt wird, sondern kommt erst zum Einsatz, wenn dafür entschieden wurde. Die Szene muss passendes Wissen komprimierter erklären und sollte besonders performant sein. Das Ziel ist ein anderes als die Informative Lehrsitzungsszene, auch wenn es oberflächlich betrachtet dieser sehr ähnlich erscheint. Es muss schnell nur passendes Wissen darstellen, welches sowohl abhängig von dem aktuelle gezeigten Hilfswissen, dem aktuellen informativen Lehrmaterial als auch von dem Profil beeinflusst wird.
- Evaluationsszene: Diese Szene erlaubt die Verwaltung und Interaktion mit den Ansichten der Evaluation. Es ist eine unabhängige Szene mit eigenen geringen Anforderungsbereich. Dennoch sind die Daten und die Motivation des Lernende es versteht und ausfüllt sehr wichtig.

Zusatzszenen

Lernende werden auch mit anderen Szenen interagieren, diese sind aber nicht Bestandteil des Lehrprozesses, darunter fällt die Auswertung und Zusammenfassung ihres Lernfortschritts, die Einstellungen des ITS selbst und die allgemeinen Menüs. Somit haben die Einstellungen für das Programm eine Szene, die „Programmverwaltungsszene“ genannt wird und die Anzeige der Erkenntnisse über den Lernenden werden durch die üblicherweise in Lernprogrammen genutzten Namen Statistik „Statistikszene“ genannt. Darunter fällt die Auswertung des ITS des Lernenden und die Anzeige seines Lernfortschritts, häufig in Form von Statistiken, daher der Name. Auch ausgefeiltere Auswertungen wie Hinweise auf Stärken und Schwächen, Vorschläge auf was man sich konzentrieren soll, die wichtigsten vom Lernenden gemachten Fehler und wie man sie vermeidet. Es stellt die Gesamtentwicklung und -bewertung dar und soll Aufschluss über den Lernenden geben, damit die Selbsterkenntnis und -einschätzung motiviert und zielgerichteter lernen lässt. Die Rollen welche in den ITS-Szenarien gefunden wurden werden verwendet um neue Szenen auszumachen, darunter fällt der Betreuer, der Lehrer und der Autor. Alle Rollen benötigen ein speziell angepasste Benutzerschnittstelle die auf andere Aufgabe zuge-

schnitten sein muss, als für den Lernenden. Für diese Rollen wird je eine eine Szene definiert. Die „Autorenszene“ spezialisiert sich dabei auf die Editierung von großen Datenmengen, es muss einfach sein Daten einzusehen, zu verändern und zu löschen. Die „Betreuerszene“ richtet sich an betreuende Personen die, auch wenn sie nicht lokal da sind, die Arbeitsleistungen von mehreren Lernenden kontrollieren und die bei Interaktionen mit dem ITS helfen. Seine Verwaltungswerkzeuge sind auf die Lernenden fokussiert. Da sich der Betreuer gut mit dem ITS auskennen, muss weniger Wert auf die Intuitivität gelegt werden, sondern es kann stattdessen professioneller aufgebaut werden. Die „Lehrerszene“ soll gerade nicht das ITS ersetzen und will die Belastung des Lehrers gering halten, sonst könnte man ja gleich den Lehrer einsetzen. Der Lehrer soll ungebunden vom Lernenden eingreifen können, wenn das ITS versagt. Dazu muss das ITS den Kontext liefern bei dem der Lernende gerade Probleme hat das ITS zu verstehen.

Zusatzszenen beinhaltet

- Programmverwaltungsszene: Die Programmverwaltungsszene verwaltet die Ansichten welche nur der Navigation und Metainformationen dienen. Darunter fällt das Hauptmenü, oder Informationen wie Versionsnummer der Applikation. Es bestimmt was passiert wenn. Es besitzt nur eine geringe Komplexität.
- Statistikszene: Diese Szene ist für die Interaktionen der Nutzer mit der Zusammenfassung und den Statistiken über die Benutzung und Auswertung des Lernenden. Es vereint die Anzeige von Diagrammen und deren Einstellmöglichkeiten, wie zum Beispiel filtern, sortieren, Auswahl einer speziellen Lernsession, Auflistung nach bestimmten Parametern und so weiter. Es füllt die Ansichten der Szene mit Daten, zum Beispiel Vorschläge als Sätze die mit dem Nutzerprofil gewählt und angepasst wurden.
- Autorenszene: Diese Szene verwaltet die Ansichten für einen Autor, wie zum Beispiel einem „Knowledge Engineer“. Da die Ansichten speziell auf die Bearbeitung von den Wissensdatenbanken ausgelegt ist, gibt es dafür auch eine spezielle Szene, die deren Eingaben annimmt und Entscheidungen trifft was passiert und welche Funktionalität und konkrete Ansicht benötigt wird.
- Betreuungsszene: Diese Szene verwaltet die Ansichten für die Betreuer, um ihre Aufgaben zu erfüllen und beinhaltet die Entscheidungen, wenn Interaktionen zwischen Betreuer und Lernender auftreten.
- Lehrerszene: Diese ist für den Lehrer, seine Tätigkeit auszuhelfen, wenn das ITS selbst Hilfe bei der Lehre benötigt. Dem Lernenden muss nicht einmal bewusst sein, dass in wenigen Fällen ein Lehrer aushilft. Der Lehrer kann zum Beispiel auch nur Entscheidungen für schwierige Prozesse im ITS übernehmen. Welche konkrete Ansicht gewählt wird, diese zu kennen und bei Eingaben darauf zu reagieren und komplexere von der Ansicht unabhängige Funktionalität in der Funktionalschicht zu erfragen ist Aufgabe dieser Szene.

Navigationsbaum

Beispielhaft wird ein Navigationsaufbau der verschiedenen Ansichten in Abbildung R-III2e-2 gezeigt. Hierbei sieht man als Elemente die konkreten Ansichten der Interaktionsschicht eines ITS. Solch ein ITS besitzt in der Produktversion sehr viel mehr konkrete Ansichten, zum Beispiel Fehleransichten beim Anmelden, Verbindungsabbrüche, Nutzerregistrierung und verschiedenste Fenster für die verschiedenen Bereiche. Dennoch lässt sich deren Absicht in die Szenen einteilen und damit auch deren Anspruch und die benötigten Ressourcen. Zum Beispiel braucht eine Anmeldenansicht keinen Zugriff auf den Lehrprozess oder das Domänenwissen. Das vollständige Beispiel würde ausarten, das gezeigte Beispiel zeigt schon deutlich wie die verschiedenen Ansichten den Szenen und

damit den benötigten Funktionen und temporären Informationen die dafür wiederum gebraucht werden leicht zugeteilt werden können.

Das ITS, welches hinter diesen Ansichten stehen würde, bildet hier alle Rollen ab und kennt diese durch die Zuweisung eines Nutzernamens. Beim Anmelden in dem System wird der Nutzer dementsprechend automatisch in eine Szene geleitet, welche seiner Rolle entspricht. Die möglichen Nutzeraktionen für die Rolle sind begrenzt. Der Autor kann vorhandenes Wissen verändern und der Betreuer kann motivieren und die Arbeit kontrollieren oder Hilfestellungen bezüglich der Bedienung des ITS geben. Der Lehrer sieht den Ablauf des Dialogs zwischen Lernenden und ITS und kann auf Anfrage eine manuelle Hilfestellung bereit stellen. Der Lernende selbst kann die Statistik, die Einstellungen oder eine Lehrsitzung starten.

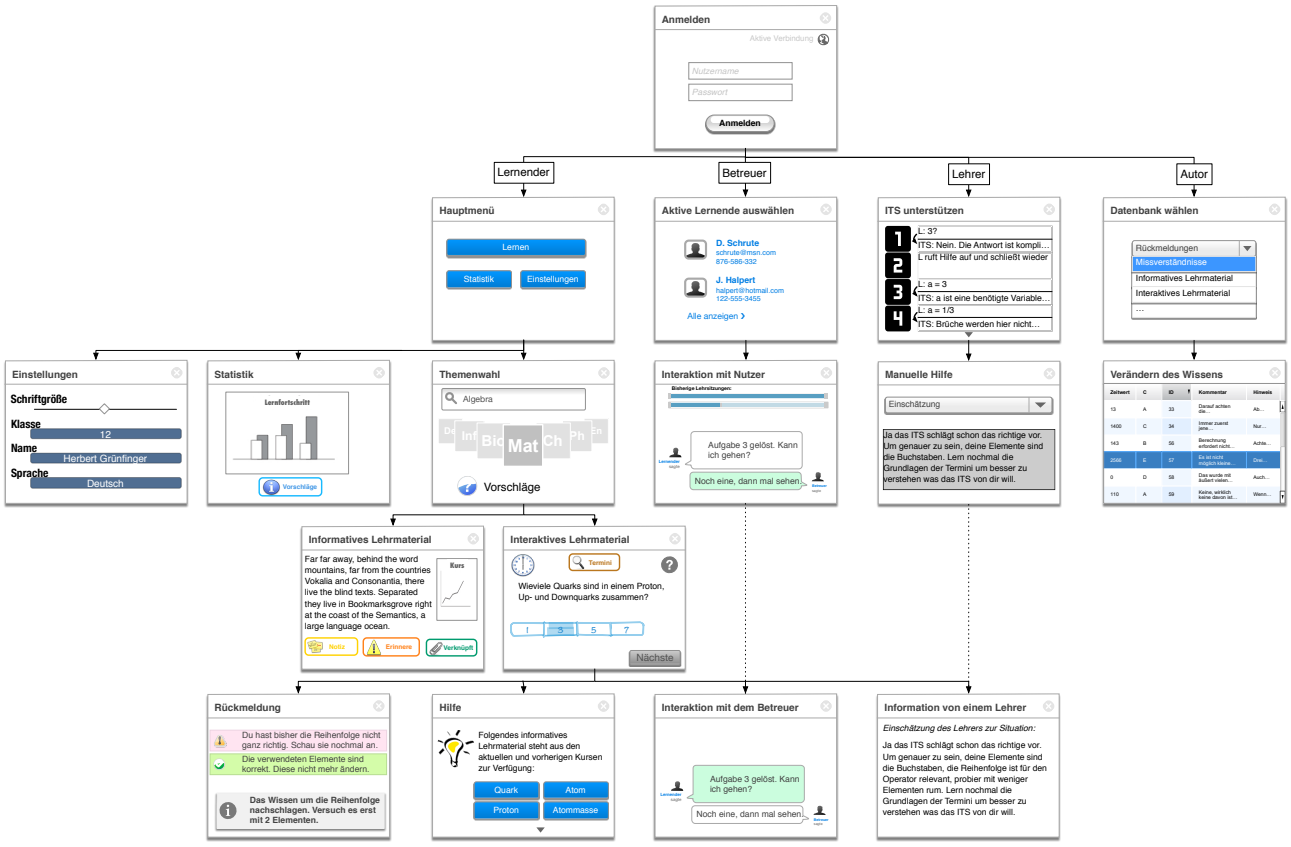


Abbildung R-III2e-2: Beispielaufbau der Ansichten in einem ITS

III.2.f. Komponenten der der Funktionalschicht im Detail

Die hier entwickelten internen Funktionskomponenten stellen die Funktionen dar, welche als Anforderung an ein ITS gefunden wurden und bei der Lehraufgabe des ITS helfen. Im Vordergrund steht die Adaption des Programms und die Interaktion mit dem Lernenden. Die Anforderungen an die Funktionen aus dem ITS-Prozess können mit der ITS-Kernfunktionalität abgebildet werden, während alle anderen Funktionalitäten mit den Programmfunktionalität abgedeckt sind. Die Schwierigkeit bei der Auswahl der Komponenten ist sie abstrakt genug zu wählen und nur die Funktionen zu nehmen die wichtig genug sind für ein ITS. Es ist schwierig diese Elemente zu identifizieren und in die richtige Detailebene zu kategorisieren. Das Ergebnis ist in Abbildung R-III2f-1 zu sehen.

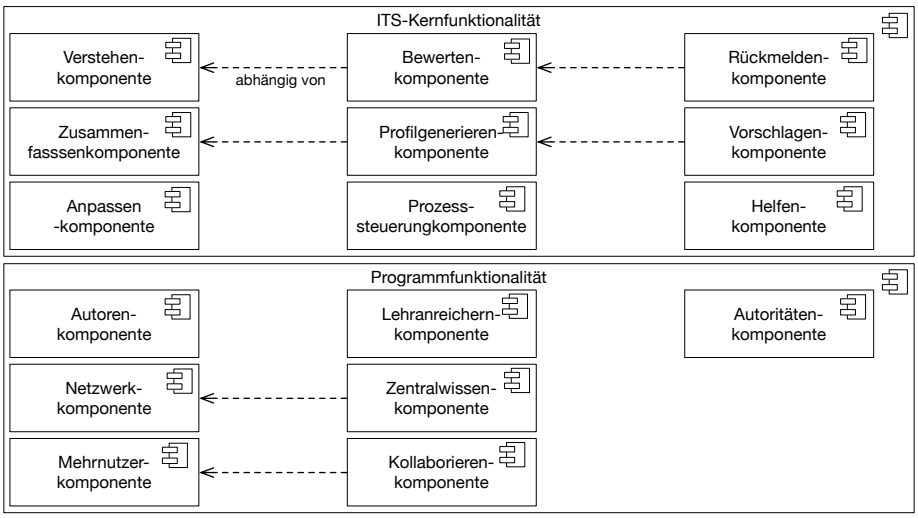


Abbildung R-III2f-1: Komponenten der Funktionalschicht in der dritten und vierten Abstraktionsebene

ITS-Kernfunktionalität

Die Anforderungen an die ITS-Kernfunktionalität lässt sich anhand des bereits definierten Prozesses analysieren. Die ersten Schritte sind die Initialisierung der Datenbanken. Diese gehören nicht zu der Interaktion mit dem Lernenden und dienen nur indirekt der Lehrschleife in dem Prozess. Genau wie die Verwaltung und Eingabe von neuem vom Lernendem unabhängigem Wissen gehört es daher auch nicht zur ITS-Kernfunktionalität. Ein anderes entscheidendes Merkmal ist, dass die Funktionalität Datenbanken zu laden eher allgemeinere Programmfunktionen darstellt.

Der Prozess selbst bekommt eine Komponente, die „Prozesssteuerungskomponente“. Es könnte der Prozess, wie in den bekannten ITS, in die jeweiligen Komponenten integriert werden, dies führt aber dazu, dass ein Teil des Gesamtprozesses immer ein Teil der jeweiligen Funktionalität darstellt. Um den Prozess auch ändern zu können, ohne die Funktionalität zu verändern, ist es notwendig, dass es seine eigene Komponente hat. Ein zusätzlicher Vorteil ist, dass der Prozess explizit ist und es damit offensichtlicher im Programmablauf ist. Das führt zu einer klaren Struktur, bei dem eine Nichteinhaltung sofort ersichtlich wird und es nicht mehr einfach wird den Prozess kompliziert in den jeweiligen Komponenten zu verstecken. Es ist also mit einer Prozesssteuerungskomponente nicht mehr nötig, sich in die unterschiedlichen Komponenten hineinzudenken nur um den Prozessablauf im Gesamten zu verstehen. Der Prozess wird damit von den Komponenten abstrahiert und erleichtert damit seine gesamte Verwaltung und Einhaltung.

Um sich an den Lernenden überhaupt anpassen zu können, müssen die gesammelten Daten über diesen erst ausgewertet werden. In einem ersten Schritt werden dabei die Daten zusammengefasst und zu weiteren Daten ausgewertet. Diese Aufgabe übernimmt die „Zusammenfassenkomponente“. Damit werden neue Daten generiert, die sowohl eigenständig kompakter benutzt werden können als auch neue Metriken verwendet, die immer aussagekräftiger werden, wenn mann mehr und mehr Daten über den Lernenden sammelt. Diese Metriken lassen sich direkt von den gesammelten Daten ableiten und dabei wird nicht geschätzt. Die Ergebnisse lassen sich einerseits für Menschen anzeigen zur eigenen Einschätzung und Analyse oder es kann dem System überlassen werden Rückschlüsse zu ziehen.

Bei dem Profil reicht es nicht aus die Lernendenwissenkomponente zu laden, da diese per Definition keine tiefgehende Analyse der Interaktionen des Lernenden übernehmen kann. Auch reicht eine weitere Auswertung der Zusammenfassenkomponente nicht aus, weil die Schlussfolgerungen fehlen, welche abhängig von der Interpretation sich stark unterscheiden kann. Es fehlt also eine Komponente, welche eine Funktionalität bereit stellt,

die genau dies macht: Die „Profilgenerierenkomponente“. Diese kann über den Lernenden anspruchsvolle Analysen über die Stärken und Schwächen des Lernenden in den Themenbereichen, Modellarten, Lehrweisen, Tageszeiten oder anderen Faktoren avancieren. Eine genaueres Profil ermöglicht eine bessere Adaptivität des ITS an den Lernenden. Ein gute Adaption lohnt sich nur dann wenn man ein gutes Profil als Grundlage hat, denn eine gute Adaption an ein ungenaues oder stark fehlerbehaftetes Profil wirkt sich negativ aus.

Bei der Generierung der Lehreinheit fehlt die Funktionalität zum Profil passende Domänenthemen herauszufiltern und dazu passendes Domänenwissen zu finden. Dazu wird das zu verwendende Profil benötigt. Diese beiden Aufgaben können gut in einer Komponente zusammengefasst werden, weil die Aufgaben nur zusammen verwendet werden und sie in ihrem Aufgabenbereich nah beieinander liegen. Wenn diese Komponente durch die Themenfindung aus dem vorhandenen Domänen schon weiß, welche Wissensrepräsentationen zu den jeweiligen Themen gehören, dann ist es leichter genau die richtigen Wissensrepräsentationen zu erlangen. Diese Komponente bekommt den Namen „Vorschlagenkomponente“. Sie ermöglicht es direkt oder nach Suchkriterien des Nutzers zusammenpassende Wissenselemente zu finden und dann nach dem vorhandenen Annotationen aus dem Pädagogikwissen zu sequenzialisieren. Nachdem vorgeschlagen wurde, sind als Ergebnis die Spezifikation des passenden Domänenwissens gefunden, sortiert und gruppiert.

Durch die Eingabe von Kriterien für die Auswahl von Domänenwissen kann das Lehrmaterial jetzt aus den Elementen des Domänenwissens, den Wissensrepräsentationen, zusammengesetzt werden. Das heißt nicht, dass alle gewählten Bausteine komplett verwendet werden müssen. Sowohl welche Bausteine gewählt werden als auch welche Teile davon präsentiert werden ist abhängig von den Gegebenheiten der Szene, des Systems und den Präferenzen und Fähigkeiten des Lernenden. Es müssen die Informationen auf ein Maß reduziert werden, welche den Umständen entsprechen, damit die letztendlich verwendete Sequenz an aneinander folgendem Lehrmaterial auch auf eine für den Lernenden aufnehmbaren Länge entspricht. Die Kriterien sind wie ein Filter und werden durch die Vorschlagenkomponente generiert, die das Domänenwissen in Gruppen sortiert hat und von der eine von einem Lernenden ausgewählt werden könnte. Die gewählten Bausteine werden sequentiell angeordnet und falls möglich könnten sogar Bindeglieder zwischen Einzelnen davon für einen geschmeidigeren Übergang sorgen. Falls das Domänenwissen Regeln für die Generierung von Lehrmaterial eines Bausteins festlegt, werden diese Regeln ausgeführt, wenn möglich mit Parametern passend für das System und den Lernenden. Für das aktive Lehrmaterial kann eine Aufgabe zum Beispiel den Schwierigkeitsgrad variieren oder nur ungelöste Aufgaben stellen. Regeln für die Generierung von Übungsaufgaben sind in formal eindeutig in Regeln basierten Domänen einfacher und erlauben eine Vielzahl an Aufgaben mit weniger Aufwand. Solch eine Funktionalität kann komplett in einer Komponente gehandhabt werden, welches das Lehrmaterial an den Einzelnen anpasst: Die „Anpassenkomponente“. Das Ergebnis ist Lehrmaterial, welches nicht speziell für eine Ansicht angepasst wurde, sondern für den Lernenden und das Lernsystem mit seinen Grenzen, zum Beispiel ein kleines Mobilgerät mit wenig Arbeitsspeicher oder einen Heimcomputer (PC). Die feinere Aufbereitung der Präsentation mit derem Layout (Größen der Elemente zueinander und dem Gesamtkontext, im Detail zum Beispiel die Schriftgröße) und festlegen welches Element wie grob angezeigt oder abgespielt werden soll, übernimmt in der klassischen Softwarearchitektur die Benutzerschnittstellenkomponente. Die konkrete Umsetzung auf einer Platform ist Aufgabe der Komponenten der Interaktionsschicht. Reicht die Benutzerschnittstellenkomponente zusammen mit der Abspeicherung des Fortschritts des gezeigten informativen Lehrmaterials in dem Lernendenwissen durch die Lernendenwissenkomponente im informativen Teil des Prozesses aus, so wird im interaktiven Teil ein Dialog des ITS mit dem Lernenden erwartet.

Die Eingaben des Lernenden sind in dem interaktiven Teil des Lehrprozesses so offen gestaltet, dass das System nicht ohne Weiteres den Sinn hinter der Eingabe semantisch greifen kann. Hier kommt die „Verstehenkomponente“ ins Spiel. Ihre Aufgabe ist es die Eingabe so zu transformieren, dass das System nur noch semantisch bekannten Konstrukten umgehen muss. Von der eigentlichen Eingabe muss nichts als der semantische Inhalt übrig bleiben. Ein Beispiel dafür ist das Aufteilen eines Eingabetextes in Schlüsselwörter durch herausfiltern von Füllwörtern in einer natürlichsprachlichen kurzen Zeichenkette.

Die Bewertenkomponente kann mehrfach eingesetzt werden. Sowohl für das informative Lehrmaterial, das interaktive Lehrmaterial, als auch die Evaluationen. Es setzt die Eingaben in den Kontext und bewertet ob es für diesen Kontext eine positive oder negative Eingabe war. Bei dem interaktiven Lehrmaterial ist der semantische Inhalt der Eingabe durch die Verstehenkomponente erlangt worden und wird noch in den Kontext gesetzt, ob dies überhaupt zielführend für die Lösung der Aufgabe ist. Nur mit einer Bewertung kann vom ITS eingeschätzt werden ob der Lernende eine gute oder schlechte Antwort gegeben hat. Dies trifft auch für die beschränkteren Eingabemöglichkeiten beim informativen Lehrmaterial zu, der den Einsatz von Hilfe, die Verweildauer auf einer Seite, das zurückgehen oder mögliche einfache Pfade bewertet. Eine Evaluation ist auch zu bewerten in Abhängigkeit von der Schwere der Situation in der der Lernende war. Darunter können alle Faktoren fallen die es für eben jenen Lernenden erschwert oder erleichtert hat, wie die Tageszeit, das Thema, die Erfolge oder Misserfolge vorher und so weiter. Die Bewertenkomponente kann reduziert werden, wenn die Eingaben weniger Freiheitsgrade besitzen und deren Einschätzung eindeutig wird. Dies geht bis hin zur bloßen Kenntnisnahme vom ITS, dass eine Eingabe getätigt wurde. Zum Beispiel wird die Eingabe des Lernenden beim Model-Tracing-Verfahren wird mit einem oder mehreren Referenzlösungen und deren Pfaden verglichen und auf Abweichungen wird hingewiesen. Beim Contraint-Based-Verfahren werden Bedingungen überprüft die eingetroffen sind.

Der zweite Teil eines Dialogs ist die Antwort des Gegenübers. Bei einem ITS kann dies eine direkte Antwort auf eine Eingabe oder ein einfaches Hinweisfeld sein, welches bei bestimmten Aktionen eingeblendet wird. Hinweise können dabei einfache Stichpunkte, Referenzierungen auf Grundlagenwissen, Musterlösungen, generierte angepasste Texte oder andere Anzeigen sein, die den Lernenden zu einer für die Aufgabe richtigen Interaktion helfen. Diese Antwort nicht statisch zu erzeugen ist sehr schwierig. Die Rückmeldung ist dann nicht nur abhängig von der Eingabe sondern auch von der aktuellen Situation der Lehrsitzung als auch vom Profil des Lernenden. Auf Basis was in der Verstehenkomponente als semantischer Inhalt der Eingabe erkannt wurde und ob diese gut oder schlecht von der Bewertenkomponente bewertet wurde, wird eine statische Rückmeldung gewählt, eine Schablone für eine Rückmeldung angepasst oder eine neue Rückmeldung anhand von Regeln generiert. Bei Abweichungen kann der Fehler mit der Fehlerdatenbank verglichen werden um schneller zu einer guten Rückmeldung zu kommen. Um eine leichtere Austauschbarkeit zu garantieren, aber vor allem weil dies eines der herausfordernden Bereiche in der Implementierung eines ITS ist, wurde die Generierung der Rückmeldung des ITS in eine eigene Funktionskomponente mit dem Namen „Rückmeldenkomponente“ getan. Dem Temporären Zustand ist bekannt, in wie weit die Lösung der Aufgabe vorangeschritten ist und die Bewertenkomponente teilt mit was die Eingabe mit den jeweilen Informationen des Lernenden und der Aufgabe des interaktiven Teils der Lehrsitzung bedeutet: Einen Lösungsschritt weiter gehen, komplett die Aufgabe löst, einen Fehler begeht und so weiter. Auf eine korrekte Lösung ist mit einer vorgefertigten Antwort einfach zu antworten. In bekannten Fehlerfällen kann eine vorhandene Antwort In anderen Fällen is eine teilweise oder komplett vorgefertigte Antwort als Reaktion auf die Eingabe und gibt diese auch sofort zurück. Bei vielen anderen Fällen muss weitaus mehr Arbeit geleistet werden.

Falls der Lernende nicht näher zur Lösung der Aufgabe kommt, gibt es für ihn die Hilfe. Die passive und aktive Hilfe sind Funktionalitäten, die unabhängig von der Rückmeldung des ITS auf den normalen funktioniert und ist deshalb eine eigene Komponente: Die „Helfenkomponente“. Sie reagiert nach einer gewissen Untätigkeit, entweder nach Zeit oder nach entdecken, dass der Lernende von selbst nicht weiter kommt. Die Entdeckung der Hilflosigkeit ist auch in ihr integriert. Eine Helfenkomponente kommt immer dann ins Spiel, wenn die Eingaben des Lernenden nicht ausreichen, damit das System diesen bestätigt oder wenigstens in einen Fortschritt hin zur Lösung erreicht. Die Helfenkomponente reagiert unabhängig von der Rückmeldenkomponente oder Eingaben und kann neue Ansätze oder benötigtes Wissen aufzeigen.

Zu guter Letzt gibt es noch die Evaluation für eine subjektive Bewertung. Diese Daten werden auch im Lernendenwissen gespeichert und können in der Auswertung in der Profilgenerierenkomponente mit in das Profil hineinfließen. Mit solchen Informationen lassen sich die Lehrmaterialen an die Wünsche des Lernenden anpassen. Auch wenn sie vorerst keinen messbaren Unterschied gemacht haben, so verbessert es womöglich die Motivation und damit langfristig die Ergebnisse des Lernenden aufgrund längerer Nutzung. Für dieses Element wird keine neue Funktionalität benötigt. Es gibt verschiedene Abhängigkeiten, die der externen und der internen Funktionskomponenten. Die Abhängigkeit der Funktionskomponenten ist dadurch gegeben, dass zwei Funktionskomponenten eine Teilfunktionalität besaßen welche alleine auch eine Funktionskomponente darstellen kann. Ob eine Teilfunktionalität eine eigene Funktionskomponente darstellt ist mit der Frage verbunden, in wie weit die neu entstehende Funktionskomponente eine eigenständige Berechtigung hat verwendet zu werden. Zusammenfassend werden nochmal alle Funktionskomponenten der ITS-Kernfunktionalität folgend aufgezählt und definiert.

ITS-Kernfunktionalität beinhaltet

- Prozesssteuerungskomponente: Die Aufgabe der Prozesssteuerungskomponente ist die Sicherstellung der Einhaltung des Lehrprozesses, sie bildet damit den groben Ablauf ab. Sie kann den nächsten Schritt für das ITS entscheiden, die für die Lehrfunktionalität relevant ist. Sie setzt unter anderem die Entscheidung zu welcher nächsten Szene gewechselt wird um. Benötigt Informationen über den groben Fortschritt des Gesamtprozesses.
- Zusammenfassenkomponente: Diese Komponente hat die Aufgabe eine Zusammenfassung des Lernenden zu erstellen. Diese wird aus dem Lernendenwissen generiert und erzeugt weitere Werte, die keine Interpretation beinhalten. Die Zusammenfassung kann entweder von einer Szene verwendet werden um die Daten anzuzeigen oder von der Profilgenerierenkomponente verwendet werden.
- Profilgenerierenkomponente: Erstellt ein Profil aus dem Lernendenwissen. In dem Nutzerprofil werden die Zusammenfassungen interpretiert und die Analyseergebnisse des Logbuchs über die Interaktionen des Lernenden im Lernendenwissen gespeichert. Sie dienen zur Generierung von Verhaltensvorschlägen oder Erfassung der stärken und schwächen des Lernenden. Dieser Bereich nutzt Erkenntnisse von „Big Data“ und „Learning Analytics“.
- Vorschlagenkomponente: Eine Komponente die abhängig von der Profilgenerierenkomponente ist. Sie macht durch das Pädagogikwissen gute Vorschläge welches Thema und welche Wissensrepräsentationen für die nächste Lehrsitzung und einzelnen Lernenden sinnvoll ist. Dies wird durch die Informationen von dem Profil abgeleitet. Benötigt Informationen über den Lernenden.
- Anpassenkomponente: Sie ist für die Anpassung des gewähltem Domänenwissens an den Lernenden zuständig. Das Wissen über den Lernenden aus dem Lernendenwissen und dem Profil wird dafür benötigt. Somit können zum Beispiel bereits bekannte Informationen ausgeblendet werden. Benötigt Liste der passenden Wis-

senselemente. Benötigt Auswahlkriterien für das Lernmaterial und Informationen über den Lernenden.

- Verstehenkomponente: Die Aufgabe dieser Komponente ist es Eingaben die nicht bereits in einem semantisch bekannten Schema entsprechen so aufzuarbeiten, dass daraus der Inhalt für das System verständlich wird. Dies ist immer dann von Nöten, wenn die Eingabe des Nutzers sich nicht auf Formulare oder vorgefertigten Auswahlmöglichkeiten beschränkt, sondern komplexere Konstrukte erlaubt wie natürliche Sprache, Bilder, Diagramme oder eine „Domain Specific Language“ (DSL). Das trifft insbesondere auf den interaktiven Teil des Lehrprozesses zu. Da hier durch die freien Interaktionen es häufig vorkommt, dass der Lernende eine unerwartete Eingabe macht. Benötigt aktuelles Profil des Lernenden aus Temporärschicht um zu ahnen was gemeint wurde.
- Bewertenkomponente: Wertet die Interaktion des Lernenden im Kontext aus. Die kleinste Bewertung ist einfach nur die Kenntnisnahme, dass etwas gemacht wurde, dass heißt die Speicherung was der Lernende gemacht hat in die Logs oder die Evaluationen speichern. Für das informativen Lehrmaterial präsentierte Wissen kann es dem Lernenden möglich sein selbst eine begrenzte Wahl für seinen Lernpfad zu unterschiedlichen Bereichen zu haben, dies geht unter anderem durch ein Hilfegesuch. Entscheidet sich der Lernende für eine Hilfe, Entscheidet sich der Lernende für die nächste Seite oder schlägt er etwas zu der aktuellen Lehrsitzung verlinktes nach. Wie lange wurde dafür gebraucht und war es lange für die aktuelle Seite. Abhängig vom Inhalt der gerade gelehrt wird ist die Bewertung für die gleiche Aktion unterschiedlich. Dies trifft auch für das interaktive Lehrmaterial und die Evaluationen zu. Es ist erforderlich das interaktive Lehrmaterial zu bewerten, weil dies eine Grundlage für den Fortschritt zur Lösung der Aufgabe als auch für die Rückmeldenkomponente notwendig ist um passende Antworten zu wählen oder zu generieren. Bei den Evaluationen ist es wiederum nicht so schwierig aber sinnvoll diese zu bewerten um daraus weitere Schlüsse auf den Lernenden und seine Eigenschaften zu ziehen. Zum Beispiel ist eine schlechte Evaluation für eine für den Lernenden schwierige Lehrsitzung anders zu verwerten als die gleiche Evaluation für eine für den Lernenden einfache oder unpassendere.
- Rückmeldenkomponente: Eine Komponente die abhängig von der Komponente Auswerten ist. Passend zu der Antwort, welche der Lerner zu einer Frage oder Inhalt gegeben hat, gibt das ITS eine direkte Rückmeldung um lobend zu wirken oder auf mögliche Missverständnisse in dem Lernprozess des Lernenden hinzuweisen. Feedback benötigt einen Zugriff auf viele Datenbanken, da es eine schwierige Aufgabe ist die mit einer guten Datenlage der Kontext einfacher zu erfassen ist.
- Helfenkomponente: Als Funktion für den Lernenden, wenn dieser aktiv nach Hilfe verlangt oder passiv Hilfe angeboten wird. Kann auch nach der Bewertenkomponente eingesetzt werden. Die Komponente schätzt die aktuelle Situation ein, indem es das Verhalten des Lernenden mittels des vorhandenen Profils in der Temporärschicht und der aktuellen Lehrsitzung auswertet. Falls eine gewisse Zeit verstrichen ist oder durch andere Aktionen eine Hilfe erforderlich erscheint wird die Hilfe erstellt. Passt nicht das Lehrmaterial an, sondern findet optimales Lehrmaterial und bestimmt wann und wie stark Hilfe benötigt wird. Benötigt letzte Interaktionen des Lernenden und Profil über den Lernenden aus der Temporärschicht.

Programmfunktionalität

Neben den ITS-Kernfunktionalität gibt es auch die Programmfunktionalität. Diese besteht aus funktionalen Komponenten mit Aufgaben allgemeinerer Natur als die ITS-Kernfunktionalität und nicht zwingend für die Lehrfunktion eines ITS. Sie sind zwar optional, nichtsdestotrotz sind sie für ein ITS relevant, weil sie wichtige Funktionen für ein ITS bie-

ten, die in vielerlei Hinsicht der Lehre helfen. Schließlich ist es die adaptive Lehrfunktion, die ein ITS definiert. Die Komponenten der Programmfunktionalität lassen sich daher mit einer Ausnahme nicht aus dem ITS-Lehrprozess ableiten, sondern werden hauptsächlich den Anwendungsfällen entnommen. Es bleibt noch der Teil des ITS-Lehrprozesses, der die Initialisierung des Domänen- und Pädagogikwissens betrifft, welcher in die Architektur aufgenommen werden könnte. Das Domänenwissen muss darin als ersten Schritt initialisiert werden. Doch bevor es überhaupt initialisiert werden kann muss es existieren. Von Domänenexperten kann nicht verlangt werden, dass sie komplizierte Datenbanksprachen und die genaue Syntax und Semantik der Datenbankfelder der jeweiligen ITSs kennen mit denen sie aktuelle arbeiten. Deshalb muss für die Langlebigkeit eines ITS dieses über eine Funktionalität verfügen, die es ermöglicht, vereinfachte Eingabemasken zu verifizieren und umzuwandeln. Die Art uns Weise wie Domänenexperten Wissen notieren und die Art wie es gespeichert wird kann sich abhängig von der Interpretation stark unterscheiden. Es ist damit für das Erstellen, Ändern und Löschen des statischen Lehrmaterials zuständig. Besonders bei nichttechnischen Domänen wird diese Diskrepanz stärker ausfallen und damit auch intuitive Benutzeroberflächen wertvoller. Diese Arbeit übernimmt dann die „Autorenkomponente“.

Der Zugriff und die Validierung der benötigten Daten als ersten Schritt des ITS-Lehrprozesses, werden in der Persistierungsschicht und Datenerhaltungsschicht ausreichend behandelt. Das Pädagogikwissen kann zwar genauso behandelt werden, aber nur wenn man das Pädagogikwissen nicht automatisch durch Lehranreicherungsregeln generiert. Da die Lehranreicherungsregeln nur der Erzeugung von Pädagogikwissen dient, kann diese konkret auch in ihrer eigenen Datenbank gespeichert werden und abstrahiert in der Persistierungsschicht durch die Pädagogikwissenkomponente verwaltet werden. Die Funktionalität die Lehranreicherungsregeln zu generieren kann in der Persistierungsschicht nicht abgebildet werden, weil es den Aufgabenbereich davon verlässt. Es hat nicht die Aufgabe neues Lehrmaterial zu generieren, dass ist Aufgabe der Anpassungskomponente. Es hat auch nicht die Aufgabe neues Rückmeldung zu generieren, dass ist die Aufgabe von der Rückmeldenkomponente. Also ist dies die einzige Programmfunktionalität die auf dem Lehrprozess aufbaut eine zum automatischen erzeugen von pädagogischen Wissen aus Lehranreicherungsregeln, der Name wird gewählt als „Lehranreichernkomponente“.

In den Anwendungsfällen gibt es die Möglichkeit örtlich getrennte ITS zu verwenden, dies führt zu einer Komponente welche die Onlinefunktionen des ITS verwaltet um eine stabile Kommunikation zwischen ITS und Lernenden zu haben. Auch ohne die Onlinefunktionalität müssen mehrere Nutzer miteinander kommunizieren können, seien es Autoritäten oder Lernende. Da jeder Nutzer auch eigenständig agieren können muss und diese Aktionen dem jeweiligen zugeordnet werden muss, gibt es für jeden Nutzer ein eigenes Endgerät. Diese Endgeräte müssen auch ohne eine Internetverbindung miteinander kommunizieren. Die Verwaltung von Netzwerkübertragungen lassen sich gut in eine Komponente bündeln, diese wird passenderweise „Netzwerkkomponente“ genannt.

Eine zentrale Stelle zu haben, bei dem die Wissensarten gebündelt verfügbar sind, würde die Wiederverwendbarkeit drastisch erhöhen. Die modulare Architektur erlaubt es einzelne Wissenskomponenten auszutauschen, ohne dabei die anderen Komponenten ändern zu müssen. Das Wissen ist hierbei der Schwerpunkt für die Wiederverwendbarkeit, weil dieses wesentlich einfacher auf eine neue Platform übertragen werden kann. Ein besonderes Augenmerk hat hierbei neben dem Domänenwissen und dem Pädagogikwissen das Lernendenwissen, weil die Lehrforschung damit Rückschlüsse auf Lehrstrategien und die Effizienz von bestimmten Aspekten ziehen kann, die über ITSs im Gesamten hinausgehen. Eine Funktion für das Bereitstellen von den pseudonymisierten Rohdaten der Lernendeninteraktion als auch der anderen Wissensdatenbanken auf einen zentralen Server

würde weitereAuswertung auch noch lange Zeit nach Benutzung erlauben. Diese Funktionalität wird in der „Zentralwissenkomponente“ realisiert.

Es wurden die Beteiligten Akteure von den Anwendungsfällen untersucht und dabei gab es parallel zu dem ITS die selbstständig lehrende Rolle des Lehrers und die mit dem ITS zusammenarbeitende Rolle des Betreuers. Bei der in einem vorherigen Kapitel definieren Lehrerrolle ist es nicht notwenig Funktionen in das ITS zu implementieren, weil dieser höchstens abwechselnd mit dem ITS in einem Blended-Learning-Stil unterrichtet. Ein Betreuer allerdings ist für den Lehrauftrag auf das ITS angewiesen. Die Kommunikation mit den Autoritätspersonen Lehrer und Betreuer kann dabei nicht nur lokal sondern auch distanziert geschehen und sollte sich dabei nahtlos in das ITS integrieren. Die Benutzerfreundlichkeit wird erhöht, das ITS hat damit Wissen über die Kommunikation und die Autoritäten und der Lernende können direkt den Inhalt des ITS referenzieren. Das geht sowohl zeitversetzt als auch nahezu zeitgleich. Zeitgleich zu kommunizieren erfordert mehr Aufwand in der Umsetzung des ITS, ist aber nicht abgekapselt genug in der Funktionalität um eine eigene Komponente daraus zu kreieren. Ein Betreuer hat die Macht über Teile des Lernendenwissens, da damit sowohl Fehler vom ITS korrigiert werden können als auch Einsichten in den Lernverlauf notwendig sind. Es ergibt sich eine „Autoritätenkomponente“.

Weitere Anwendungsfälle ergeben sich durch die Erhöhung der Anzahl der Lernenden die entweder einzeln oder gemeinsam eine Aufgabe mit einem ITS lösen können. Mehrere Nutzer zu unterstützen ist eine Funktionalität, die unabhängig von der Kollaboration gesehen werden kann und bekommt eine eigene Komponente „Mehrnutzerkomponente“.

Die Kollaboration ist schon herausfordernder und is deshalb auch eine eigene Komponente „Kollaborierenkomponente“. Sie ermöglicht dass zusammenarbeiten von Lernenden in einer Lehrsitzung. Für das informative Lehrmaterial kann gewartet werden, bis alle Beteiligten es als verstanden erklärt haben. Für das interaktive Lehrmaterial ist dies jedoch komplizierter. Das ITS muss nicht nur den Lösungszustand der Aufgabe kennen, sondern auch schätzen wie das Verständnis der beteiligten Lernenden ist, auch wenn sie sich weniger beteiligt haben. Es wird dann unter Umständen nur die Beiträge gewertet die von jedem einzelnen kamen. Zudem muss für alle Beteiligten ersichtlich sein wer welche Eingaben gemacht hat.

Programmfunktionalität beinhaltet

- Autorenkomponente: Es existiert mit diese Komponente eine Möglichkeit, die Funktionen die für die Domäne angepassten Editierung des vorhandenen Wissen. Die Komponente hat damit Zugriff auf das Domänenwissen und Pädagogikwissen.
- Lehranreichernkomponente: Als eine komplexe Komponente, kann sie an Dömanenwissen gebundenes Pädagogikwissen anhand von Lehranreicherungsregeln generieren. Diese ermöglichen eine Automatisierung der Lehrmaterialerzeugung die zukünftig immer wichtiger werden wird.
- Netzwerkkomponente: Das System ist über eine Netzwerkverbindung abrufbar. Dies bedeutet, dass Lernende sich nicht am gleichen Standort befinden müssen wie das eingesetzte ITS, einer Autorität oder anderen Lernenden. Die Funktionen dieser Komponente kümmert auch sich um eine unauffällige Einbettung der Onlineverbindung.
- Zentralwissenkomponente: Diese Komponente kann Daten aus dem Lernendenwissen für die Nutzung in der Forschung bereit stellen oder hochladen. Eine Zusammenführung von gewissen Daten des Domänenwissens und dem Lernendenwissen kann angebracht sein, wie zum Beispiel das Thema des Lehrmaterials.
- Autoritätenkomponente: Der Lerner wird von einer anderen Person betreut, dies kann ein Lehrer sein oder auch nur jemand der in technischen Fragen hilft, es ist nicht zwingend erforderlich, dass die Lerner gleichzeitig anwesend sein müssen. Die

Funktionalität umfasst die Einsicht, Kontrolle und Hilfe des Lernenden ohne das Lernen an sich.

- Mehrnutzerkomponente: Es ist möglich mehr als einen Nutzer für das ITS zu haben. Diese Komponente bietet alle Funktionen an die benötigt werden um mehrere Nutzer innerhalb von einem ITS zu unterscheiden und auch nur dem aktuellen Lernfortschritt und Personalisierung zu akkreditieren.
- Kollaborierenkomponente: Für eine Zusammenarbeit in einer Lehrsitzungen zwischen Lernenden werden die Funktionalitäten dieser Komponente benötigt. Es muss dabei keine gleichzeitige Anwesenheit aller Lernenden erzwungen werden. Sie trennt welche Aktionen vom lokalen Lernenden kommen und wieviel der Lösung schon von anderen Lernenden vorgelegt wurden. Sie bereitet damit auf eine angepasste Bewertung vor, denn der lokale Lernende kann eine Lösung anbieten dessen Vorarbeit von einem anderen Lernenden kommt. Zudem passt die Komponente auf dass kein Lernender vernachlässigt wird. Benötigt aktuellen Lehrsitzungsverlauf aus der Temporärschicht.

III.2.g. Komponenten der Persistenzschicht im Detail

Es wurde schon in dem vorherigem Kapitel bemerkt, dass die Komponenten der Persistierungsschicht noch zu allgemein gehalten sind. Deshalb wird eine genauere Beschreibung und Aufteilung jeweils innerhalb der großen drei klassischen Komponenten hier vorgenommen, Abbildung R-III2g-1. Es ist wichtig das die Rollen der Lehrakteure (Autor, Betreuer und Lehrer), wie sie in den Szenarien beschrieben wurden, bei Zugriffen auf die Datenbanken eine Art der Synchronisierung zu den Datenbanken mit den Lernenden haben. Ob das durch ein einklinken der Lehrakteure mit einem eigenen Werkzeug oder durch eine Kopie der Datenbanken passiert ist implementierungsspezifisch. Wichtig ist nur, dass ein Autor das Pädagogik- und Domänenwissen nicht nur für seine Ausprägung des ITS macht, sondern für die Lernenden. So auch für die Betreuer und Lehrer, die keine Einsicht in die Daten eines lokalen Lernenden benötigt, sondern die des Lernenden dem sie gerade helfen. In der Implementierung mag es damit mehrere verschiedene Versionen der Datenbanken geben, die womöglich jeweils auf anderen Computern gespeichert, gestreamt, zwischengespeichert oder ähnliches werden, doch abstrahiert gibt es in der Persistenzschicht nur eine Datenbank auf die sowohl von einem oder mehreren Lehrakteuren, dem Lernenden und dem ITS zugegriffen wird. Bei Autoritätspersonen, die Einsicht in eine kollaborativ arbeitende Gruppe fordern, muss es möglich sein auf alle Lernenden einzeln Zugriff zu haben und sich zur Laufzeit einen Lernenden auszuwählen oder zu wechseln.

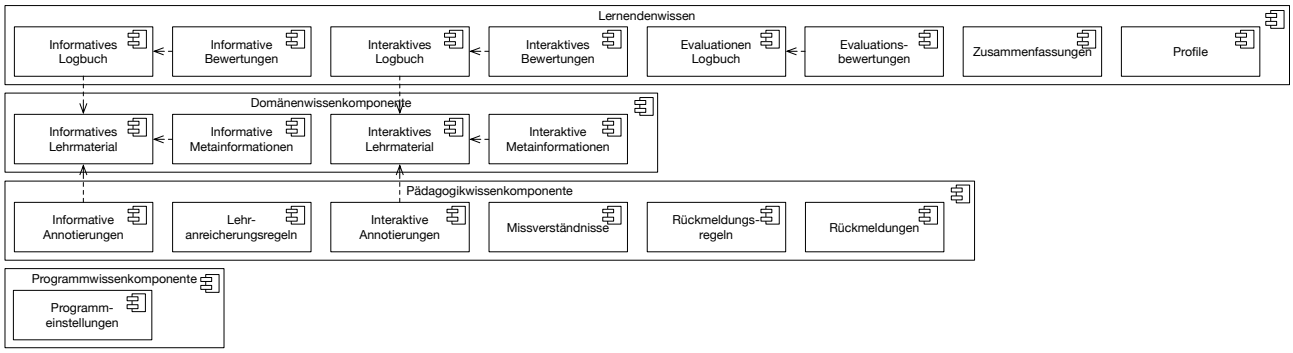


Abbildung R-III2g-1: Komponenten der Persistenzschicht in dritter und vierter Abstraktionsebene

Domänenwissenkomponente

Die Domänenwissenkomponente hat die Aufgabe das Wissen zu speichern, welches der Lernende lernen soll. Sie kann in die Bereiche aufgeteilt werden, die für das informative oder das interaktive Lehrmaterial verwendet werden. Interaktives Lehrmaterial erwartet im Gegensatz zum informativen, dass ein Input vom Lernenden erwartet wird. Es wird durch die Verkleinerung des einzelnen Lehrmaterials nicht davon ausgegangen, dass ein Datenpunkt aus einem der beiden Bereiche ausreicht um den kompletten informativen oder interaktiven Teil ein Lehrsitzung zu füllen. Die für die Lehranreicherungsregeln benötigten Metadaten für jeweils diese beiden Bereiche befinden sich auch in der Wissenskomponente. Da sich die Metadaten als dem Domänenwissen errechnen oder leicht erkennen lassen, gehört es auch dazu. Auch wenn es unüblich ist diese zu lernen, so ist es eine Information die der Lernende implizit weiß. Als Extrembeispiel zu nennen ist die Anzahl der Wörter konkret für ein Thema zu kennen unwahrscheinlich, dennoch weiß jemand ungefähr wieviele Seiten der Inhalt eines Lehrstoffs in einer gewissen Detailebene benötigt. Die Domänenwissenkomponente wird durch einen Nutzer in der Autorenrolle verwaltet und von ihm wird auch neues Lehrmaterial hinzugefügt.

Domänenwissenkomponente beinhaltet

- Informatives Lehrmaterial: Vorhandene Wissenselemente/Lehrmaterial welches in der informativen Lehrsitzung verwendet werden kann, wie Texte und Bilder.
- Informative Metadaten: Metadaten des Informativen Lehrmaterials, sie sind nicht direkter Unterrichtsstoff.
- Interaktives Lehrmaterial: Vorhandene Wissenselemente/Lehrmaterial welches im interaktiven Unterrichtsteil verwendet werden kann, wie Übungsaufgaben mit Lösung oder Regeln zum erstellen von solchen.
- Interaktive Metadaten: Metadaten des interaktiven Lehrmaterials, sie sind nicht direkter Unterrichtsstoff.

Pädagogikwissenskomponente

Die Pädagogikwissenskomponente hat die Aufgabe Daten bereit zu stellen, die dabei helfen das Domänenwissen zu lehren ohne dabei Daten über den Lernenden zu kennen. Darunter fallen einmal die Lehranreicherungsregeln für die automatische Generierung von Pädagogikwissen das an Domänenwissen gebunden ist und das für die unter den beiden Lehrmaterialtypen differenzierte Wissen. Es wird in den Anforderungen (und in den damit zusammenhängenden Papieren) erwähnt, dass die automatische Generierung von Lehrmaterial eines der Aufgaben von ITS sein müsste. Selten ist das der Fall. In Kapitel „Automatische Generierung von Lehrmaterial“ wird dazu ein Ansatz geboten wie das funktionieren könnte. Aufbauend auf dem vorgestellten Konzept werden Teile der detaillierten Datenbankstruktur entworfen. Lehrmaterialannotierungen bestehen aus Wissen über das vorhandene Domänenwissen, welches für die richtige Auswahl und Anpassung des Lehrmaterials an den Lernenden dient, als auch die hilft verknüpftem Wissen während einer Lehrsitzung zu finden, die Lehrmaterialannotierungen sind das Metadaten wie es im Kapitel „Automatische Generierung von Lehrmaterial“ gefordert wird. Interaktives Lehrmaterial hat eine durch seinen Dialog mit dem Lernenden zusätzliche Anforderungen. Fehlinterpretationen des Lehrmaterials vom Lernenden werden im interaktivem Teil der Lehrsitzung aufgedeckt. Häufige, wichtige, typische oder schon bekannte Missverständnisse sollten dem ITS schon im Vorfeld bekannt sein. Dadurch kann bei solch einer Situation ein bestimmtes effektives Verhalten vom ITS wie ein darauf spezialisierter Vorschlag oder Erklärung dem Lernenden früh helfen, dass das Domänenwissen korrekt verstanden wurde. Die Rückmeldungen vom ITS an den Lernenden sind in einem Dialog sehr relevant. Bei bekannten Variablen, kann eine teilweise oder ganz vorgefertigte Rückmeldung genutzt

werden, ansonsten muss das ITS eine Rückmeldung anhand von Regeln erstellen, welche die Intention am besten vermittelt.

Pädagogikwissenkomponente beinhaltet

- Lehranreicherungsregeln: Regeln zum Generieren von an die Domäne gebundenem Pädagogikwissen, also zum Beispiel die konkreten Lehrmaterialannotierungen zu konkretem Lehrmaterial.
- Informative Annotierungen: Annotierungen des informativen Lehrmaterials die ITSs und Lehrern hilft das Lehrmaterial auszuwählen und anzupassen: Schwierigkeitsgrad, Verbindungen zu anderen Themenkomplexen, vorausgesetztes Wissen, Basiswissen für welches andere Wissen, Bevorzugt zu verwenden für bestimmte Lernertypen, durchschnittliche Dauer für das Durcharbeiten, Besonders gut zusammen bekannten anderem Lernmaterial lehrbar und so weiter.
- Interaktive Annotierungen: Annotierungen des interaktiven Lehrmaterials die ITSs und Lehrern hilft das Lehrmaterial auszuwählen und anzupassen: Es sind sehr ähnliche Attribute wie die von dem informativen Lehrmaterialannotierungen.
- Missverständnisse: Bekannte Missverständnisse die Lernenden über den Lehrstoff haben, besonders die, welche durch Erkennen des ITS schneller korrigiert werden können
- Rückmeldungsregeln: Regeln für das Anpassen oder generieren von Rückmeldungen vom ITS innerhalb der Lehrsitzung mit dem interaktiven Lehrmaterial.
- Rückmeldungen: Vorgefertigte Rückmeldungen, welche für den interaktiven Teil der Lehrsitzung benötigt werden, die in verwendet werden können wenn eine statische Antwort passend ist. Auch vorgefertigte Lückentexte die gefüllt werden können, sind hier enthalten.

Lernendenwissenkomponente

Die Lernendenwissenkomponente soll der Fortschritt, Entwicklung und Präferenzen des Lernenden erfassen. Das kann sie im durch die penibel aufgezeichneten Interaktionen des Lernenden mit dem ITS. Dies kann wie im Domänenwissen aufgeteilt werden. Sowohl in die Interaktionen im informativen als auch in den interaktiven Teil der Lehrsitzung. Diese Aufzeichnungen werden innerhalb der Lehrsitzung bewertet um abhängig vom Kontext später besser zu verstehen und vergleichen/gruppieren zu können. Die Bewertungen werden jeweils für die Logbücher und Evaluation einzeln gespeichert. Weitere Daten über den Lernenden werden aus diesen Logbüchern und der Erfassung seiner Evaluation von Lehrsitzungen generiert. Darunter fallen die Zusammenfassungen eben jener und die wiederum daraus generierten Profile. Die Zusammenfassungen der Aufzeichnungen enthalten weitere Metriken, die nicht durch Interpretation entstanden sind, sondern durch mathematische objektive Zusammenhänge wie Anzahl, Durchschnitt, Maximum, Minimum und weitere ausgeklügelte Methoden. Das Profil ist eine komplette Einschätzung des Lernenden in den für das ITS relevanten Attributen. Die Zusammenfassungen und Profile müssen nicht gespeichert werden, weil diese aus den Rohdaten neu gewonnen werden können. Trotzdem ist dies aufwändig und eine Speicherung für zu einer besseren Performanz und einfachen Vergleich des Fortschritts des Lernenden über einen längeren Zeitraum. Die Lernendenwissenkomponente wird durch die Eingaben vom Lernenden befüllt. Dies geschieht im Gegensatz zu den anderen Komponenten aber ohne weiteres Zutun des Lernenden. Die Eingaben gehen direkt in die Logbücher und Analysen dieser Eingaben führen zu den Bewertungen, den Zusammenfassungen und den Profilen.

Lernendenwissenkomponente beinhaltet

- Informatives Logbuch: Interaktionen des Lernenden während des informativen Teils der begonnenen Lehrsitzungen
- Informative Bewertungen: Die Einschätzung der Interaktionen innerhalb des informativen Teils der Lehrsitzungen im Kontext der jeweiligen Lehrsitzung.
- Interaktives Logbuch: Interaktionen des Lernenden während des interaktiven Teils der begonnenen Lehrsitzungen
- Interaktive Bewertungen: Auswertung des ITS für Lösungsschritte/Eingaben des Lernenden innerhalb des interaktiven Lehrmaterials in einer Lehrsitzung.
- Evaluationen: Evaluationen des Lernenden über die Lehrsitzungen
- Evaluationen Bewertungen: Die Einschätzungen des Systems der jeweiligen Evaluation in der dazu zugehörigen Lehrsitzung. Eine Evaluation hängt nicht nur von den Antworten, sondern auch davon ab wie der Lernende abgeschnitten hat und wie passend die Lehrsitzung für den Lernenden war in Zeit, Materialtyp, Schwierigkeitsgrad, Thema und so weiter.
- Zusammenfassungen: Gespeicherte weitere Auswertungen der Logbücher nach objektiven Maßstäben
- Profile: Vorherige ausgewertete Logbücher und Evaluationen zu einer Zusammenfassung und/oder Analyse

Programmwissenkomponente

Die letzte Datenbank wird nicht weiter aufgeteilt, da sie sehr spezifisch für das jeweilige Programm ist und es bei der Entwicklung der Komponenten eines ITS vermieden werden soll, andere Teile des Programms zu spezifizieren. Einige Daten in einem ITS passen aber nicht in die bisher vorgefertigten Kategorien. Viele Einstellungen wie die Schriftgröße und Farbe für die Darstellung könnten durch Didaktiker festgelegt werden, diese müssen allerdings auch auf Informationen zugreifen wie eine Sehbehinderung. Zusätzlich Funktionen wie zum Beispiel die Nutzung des ITS von mehreren Lernern unabhängig voneinander: Ein Mehrnutzersystem. So ist ein ITS ursprünglich nicht für mehrere Lernende ausgelegt, man hat also keinen Ort um Nutzernamen und Passwörter zu speichern. Zudem gibt es Einstellungen, die ein Lernender in der Metaebene vornehmen möchte, die auch irrelevant für ein ITS scheinen, ob zum Beispiel Mitteilungen von dem ITS auch dann angezeigt werden dürfen, wenn es gerade nicht aktiv ist oder ob das Mobilfunkdatenvolumen verbraucht werden darf oder nur im Netzwerk eine Onlineverbindung bestehen darf. Es sollten keine großen Datenmengen gespeichert werden, sondern nur kleinere Einstellungen. Daher auch der Name: „Programmeinstellungen“. Dieses besitzt vorgefertigte Standardeinstellungen, die von einem Nutzer verändert werden können, damit wird in den allermeisten Fällen die Anzahl der Optionen fest bleiben.

Programmwissenkomponente beinhaltet

- Programmeinstellungen: Einstellungen die das Programm speichern muss, die nicht in die anderen Bereiche passen. Diese sollten gering gehalten werden.

III.2.h. Softwarearchitektur bis Abstraktionsebene 4

Alle bisher genannten Architekturelemente sind nun definiert. Diese lassen sich in vier folgend genannten Abstraktionsebenen zuordnen.

- Programmeinteilung in Schichten
- Bereiche in der jeweiligen Schicht
- Groben Bereichen
- Komponenten

Einige Teile bedingen sich nicht nur durch den definierten ITS-Lehrprozess, sondern benötigten immer Funktionalitäten von anderen Teilen. Es ist klar, dass das ITS-Programm offensichtlich Menschen benötigt um zu funktionieren, der Nutzer kann dabei einer der definierten Rollen erfüllen. Zusätzlich können nur nebeneinander liegende Schichten durch die Schichtarchitektur definitionsgemäß voneinander abhängig sein. Die Schichten können nur mit der nächstanliegenden Schicht kommunizieren. Dabei sind die oberen Schichten abhängig von den darunterliegenden, weil die tieferliegenden Schichten auch ohne die darüberliegenden funktioniert, umgekehrt aber nicht. Die Abhängigkeiten in den einzelnen Schichten wurden in der jeweiligen Tiefe in vorherigen Kapiteln definiert. Insgesamt führt das zu einer Konstellation wie sie in Abbildung R-III2h-1 zu sehen ist.

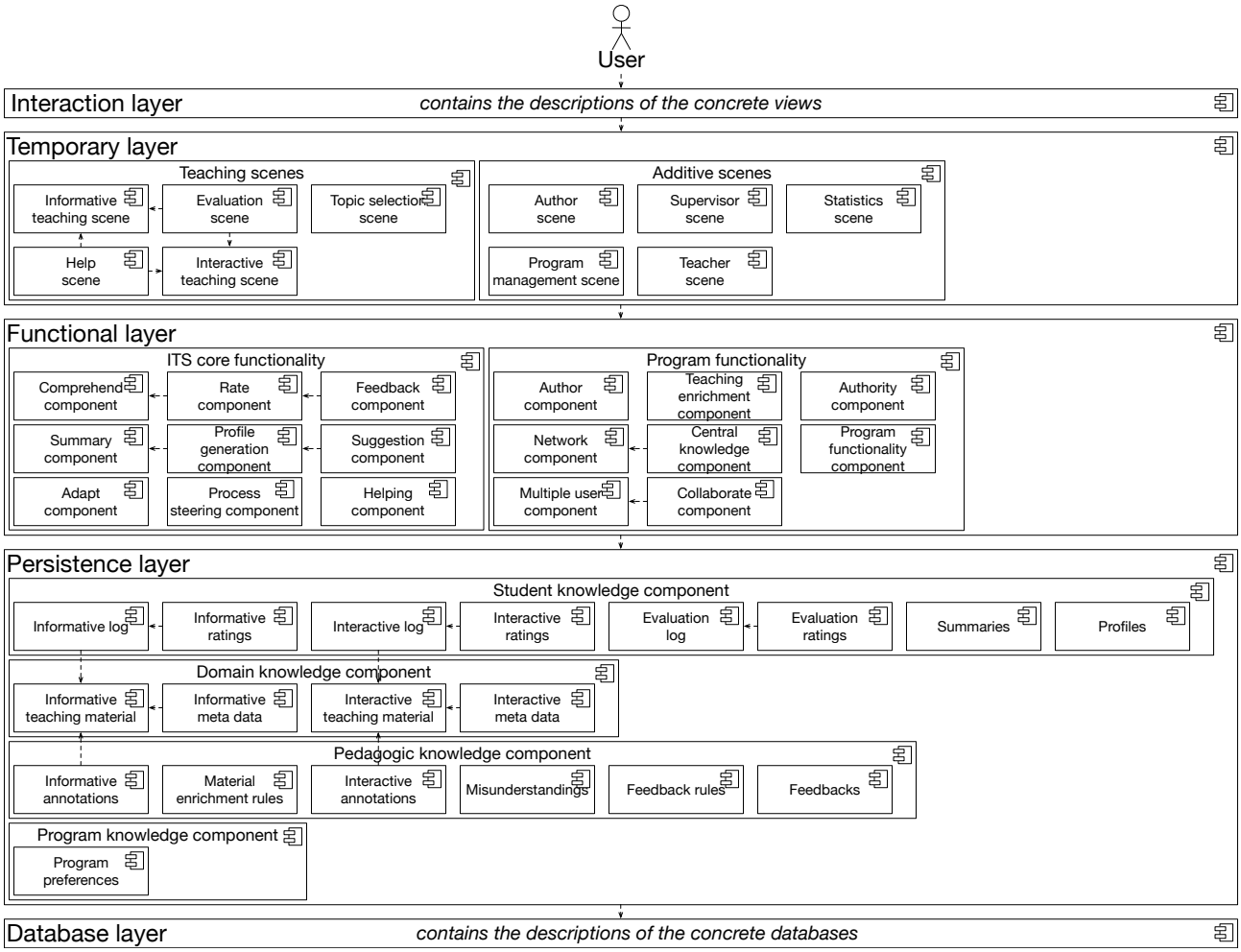


Abbildung R-III2h-1: Architekturkomponenten und deren Abhängigkeiten innerhalb der gleichen Komponente [2020a GRAF VON MALOTKY and MARTENS]

Für eine Architektur wollen wir jetzt noch die erlaubten Schnittstellen definieren, also welche Komponenten miteinander kommunizieren dürfen. Schnittstellen sind nur zwischen den übereinanderliegenden Schichten erlaubt. Die kleinsten Komponenten in diesen sollten nur mit bestimmten Komponenten der anderen Schicht kommunizieren dürfen. Da in den äußeren beiden Schichten, darunter fällt die Interaktionsschicht und die Datenerhaltungsschicht, keine Komponenten definiert wurden, kann auch keine Schnittstelle zu einzelnen kleinsten Komponenten definiert werden. Es ist aber für eine Implementierung eines ITS notwendig, dass in diesen äußeren Schichten Komponenten vorhanden sind, denn jede der Komponenten in der Persistenzschicht muss auf eine konkrete Datenbank zugreifen und jede der Szenen in der Temporärschicht muss mit einer Ansicht ausgeprägt werden. Deshalb wird dies abstrahiert als eine Schnittstelle zu der gesamten Schicht abgebildet. Alle Komponenten der Temporärschicht haben eine Schnittstelle zur Interakti-

onsschicht und alle Komponenten der Persistenzschicht haben eine Schnittstelle zur Datenerhaltungsschicht. Die Schnittstellen zur Funktionalschicht hingegen können detaillierter definiert werden, da in den inneren drei Schichten alle Komponenten definiert sind. Erst werden die Gründe für die Schnittstellen zwischen der Temporärschicht zur Funktionalschicht und dann die von der Funktionalschicht zur Persistenzschicht beschrieben. Diese Gründe bekommt man durch das Durchgehen aller Abläufe aller Interaktionen und erwarteten Reaktionen des ITS innerhalb des ITS-Lehrprozesses und der Zusatzfunktionen. Zuvor haben wir beschrieben wie wir diese Abläufe bekommen und einen idealtypische Nutzung des ITS von einem Lernenden betrachtet. Dies lässt sich auf alle Rollen und alle Situationen ausweiten. Situationen die als nicht sinnvoll betrachtet und die der einzige Grund für eine Schnittstelle sind führen zu einem entfernen der Schnittstelle. Da wir die Bedienung der Szenen aufgrund ihrer abstrakten Art abschätzen können ohne eine konkrete Version in einer Ansicht vor uns zu haben ist es möglich alle Interaktionsmöglichkeiten durchzugehen und dabei nur die notwendigen Komponenten der Funktionalschicht herauszufiltern. Im Zweiten Schritt können wir durch unsere Definition der Komponenten der Funktionalschicht und der Persistenzschicht die notwendigen Schnittstellen herausfinden. Zusammen gelangen wir zu einer Vollständigen Architektur wie sie in Abbildung R-III2k-2 zu sehen ist, in ihr gibt es für eine vereinfachte Darstellung eine Zusammenfassung aller Schnittstellen für eine Komponente mittels eines Schnittstellensymbols. Da die Abläufe benutzerbezogen sind, werden die Schnittstellenbeschreibungen aus Richtung des Nutzers eingeteilt: Die vorhandenen Komponenten der Temporärschicht teilen die Beschreibungen für Schnittstellen zwischen Temporärschicht zur Funktionalschicht und die Komponenten der Funktionalschicht teilen die Beschreibungen zwischen Funktionalschicht und Persistenzschicht auf. Auch wird nicht näher darauf eingegangen ob es nur lesender oder lesender und schreibender Zugriff auf die Daten gibt wenn es eine Schnittstelle zwischen einer Komponente der Funktionalschicht und der Persistenzschicht gibt. Es wird damit nur erlaubt, dass es eine Schnittstelle geben kann, nicht wie diese im Detail auszusehen hat.

Durch den Aufbau der Architektur und Definition der Komponenten sind Schnittstellen zwischen den Komponenten auf der gleichen Schicht nicht notwendig. Zwar kann eine Szene an eine andere den temporären Zustand übergeben, aber dies bleibt einer generischen Transitionsfunktion und es ergeben sich dabei keine Aufrufe von einer Szene in einer anderen. Die Kommunikation zwischen Akteuren aus den Szenarien wird durch die Netzwerkkomponente geregelt, da jeder Nutzer jeweils seine eigene Systeminstanz benutzt. Es muss bei der Definition der Schnittstellen von der Funktionalschicht zur Persistenzschicht immer die Frage gestellt werden, ob die Daten aus der Persistenzschicht benötigt werden oder ob es Daten aus der Temporärschicht ausreichen. Im Allgemeinen werden die Zugriffszeiten zur Persistenzschicht schlechter sein, weil die Menge an Daten größer ist und bisher gar nicht bis wenig benutzt und damit geladen wurden. Ein gutes Beispiel hierfür ist das Zugriff auf das Profil. Dieses wird von einigen Komponenten benötigt, dennoch ist es nicht notwendig Zugriff auf alle bisher gespeicherten Profile zu haben, wenn das zuletzt generierte ausreicht und geladen wurde.

Das Ergebnis ist in Abbildung R-III2h-2 zu sehen. Darauf folgend sind alle Schnittstellen der vierten Abstraktionsebene einzeln aufgeführt und beschrieben. Der Hinweis sei gegeben, dass es sich bei den Verknüpfungssymbolen jeweils um zwei Verbindungen handelt und aus grafischen und zur Übersichtlichkeit diese zusammengefasst wurden. Es konnte zwar abgeschätzt werden, wie der Informationsfluss zwischen den Komponenten ist und damit auch die Richtung der Verknüpfungen, aber es war nicht möglich dies über alle möglichen ITS abzubilden. Dies liegt auch daran, weil viele Komponenten eine asynchrone Rückmeldung benötigen. Der Einfachheit und um die Einsatzfähigkeit nicht einzuschränken wurden deshalb beide Richtungen in den Verknüpfungen erlaubt. Häufig lässt sich durch die Anzahl der Schnittstellen grob die Wichtigkeit einer Komponente ableiten,

davon ausgenommen werden sollte die Prozesssteuerungskomponente und die Zentralwissenkomponente. Dies stimmt insbesondere für die Persistenzschicht, weil dort die Anzahl an Verknüpfungen durch die höhere Anzahl an Komponenten in der Funktionalschicht und der Persistenzschicht viel höher ist. Die Grundlage für die Lehre ist das Lehrmaterial und dies spiegelt sich auch im Grad der Komponente wieder (Anzahl der Verknüpfungen der Komponente) mit acht für das Lehrmaterial, gefolgt von deren Annotierungen mit dem Grad sechs.

Abbildung R-III2h-2: Softwarearchitektur aller Komponenten bis zur vierten Abstraktionsebene

Temporärschicht Themenwahlszene Schnittstellen zur Funktionalschicht

Die Themenwahlszene beginnt wenn der Lernende eine Lehrsitzung beginnen möchte bis zu dem Punkt, an dem die Lehrsitzung mit dem präsentieren des informativen Lehrmaterials anfängt. Für den Lernenden gibt es dabei nicht viele mögliche Aktionen: Er muss einer der vorgeschlagenen Themen für die Lehrsitzung wählen, sich selbst eines raussuchen oder abbrechen, dies kann auch in mehreren Schritten vorhanden sein. Den Ablauf kann die Themenwahlszene über die Prozesssteuerungskomponente erfahren und auch dort die Kontrolle abgeben, wenn sie fertig ist. Um Themen vorzuschlagen muss das ITS erst die bisher über den Lernenden gesammelten Daten auswerten, diese Funktionalität besitzt die Zusammenfassenkomponente und für ausgeklügelte Analysen die Profilgenerierenkomponente. Somit steht eine aktuelle Version bereit und die Themenwahlszene kann damit angepasste Vorschläge für Lehrmaterial an den Lernenden durch die Vorschlagenkomponente machen. Wenn der Lernende ein spezielles Thema gewählt hat, dann muss dieses Lehrmaterial speziell an ihn angepasst werden, dies sind Funktionen die die Anpassenkomponente bereit stellt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Themenwahlszene und Zusammenfassenkomponente
- Themenwahlszene und Profilgenerierenkomponente
- Themenwahlszene und Vorschlagenkomponente
- Themenwahlszene und Anpassenkomponente
- Themenwahlszene und Prozesssteuerungskomponente

Temporärschicht Informative Lehrsitzungsszene Schnittstellen zur Funktionalschicht

Die Informative Lehrsitzungsszene greift auf vorgefertigtes informatives Lehrmaterial im temporären Zustand zu, zum Beispiel durch eine Übergabe von der vorherigen Szene mit Übergabe durch die Prozesssteuerungskomponente oder durch das Lehrszenenobjekt, wenn mehrere Schichten zur Implementierung verwendet werden. Dieses Material wird dann vom ITS präsentiert. Die begrenzten Interaktionen welche dem Studenten zur Verfügung stehen sind eine einfache Navigation und dem Lernenden passive oder aktive Hilfe bereitzustellen. Für Letzteres wird auf die Helfenkomponente zugegriffen. Jede Interaktion wird im Kontext bewertet. Zeigt die Aktion, dass der Lernende den Inhalt verstanden hat oder nicht, ist es eine Aktion um neues Wissen zu erlangen oder altes Wissen zu festigen und so weiter. Die kleinste Form der Bewertung ist die bloße Kenntnisnahme, dass eine Aktion gemacht wurde. Für diese Funktionen braucht es die Bewertenkomponente. Für den korrekten Ablauf informiert sich die Lehrsitzungsszene bei der Prozesssteuerungskomponente, zum Beispiel welche Szene als nächstes kommt. Für die Interaktion mit Autoritäten und anderen Lernenden wird die Netzwerkkomponente verwendet. Wenn der Lernende auch kollaborativ mit anderen Lernenden zusammenarbeitet, benötigt dies noch die Funktionen der Kollaborierenkomponente.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Informative Lehrsitzungsszene und Bewertenkomponente
- Informative Lehrsitzungsszene und Helfenkomponente
- Informative Lehrsitzungsszene und Prozesssteuerungskomponente
- Informative Lehrsitzungsszene und Netzwerk
- Informative Lehrsitzungsszene und Kollaborierienkomponente

Temporärschicht Interaktive Lehrsitzungsszene Schnittstellen zur Funktionalschicht

Die Interaktive Lehrsitzungsszene muss im Dialog mit dem Lernenden stehen um ihm bei der Lösung seiner Aufgaben zu unterstützen. Denn so können kleinere Hinweise und damit ein adaptiveres Verhalten des ITS erzeugt werden und es wird vermieden dass die

Hinweise zu viel der Lösung preisgeben. Dazu muss die Szene die Fähigkeit besitzen, die Eingaben des Lernenden zu verstehen und zu bewerten. Die Verstehenkomponente wandelt dabei Eingaben in eine interne semantische Struktur um, um sie nachträglich abhängig zur Aufgabe mittels der Bewertenkomponente zu bewerten. Um auf die Eingaben zu reagieren, wird von dem ITS eine Rückmeldung gegeben, entweder durch auswählen einer vorgefertigten Antwort oder durch generieren einer Rückmeldung. Diese Aufgaben werden von der Rückmeldenkomponente abgedeckt. Falls der Lernende keinen weiteren Lösungsschritt mittels der Rückmeldungen schafft, stellt die Helfenkomponente die Funktionen bereit mit denen man dieses erkennt und wie man darauf reagiert. Damit die Szene weiß was als nächstes gemacht wird, wird die Prozesssteuerungskomponente mit eingebunden, welche diese Antworten liefert und auch um bei Beendigung der Szene an eine andere Szene zu übergeben. Die Interaktive Lehrsitzungsszene kann auch von mehreren Lernenden (jeder mit seinem eigenen ITS) zusammen, mit Beaufsichtigung von einem Betreuer oder seltener Hilfe eines Lehrers verwendet werden. Die Kommunikation des ITS mit anderen Ausprägungen oder Systemen wird in der Netzwerkkomponente abgekapselt. Die Zusammenarbeit der Lernenden auf eine Lösung hin hat allerdings mehr spezielle Anforderungen die nicht in die Netzwerkkomponente inhaltlich passen. Diese sind in einer Kollaborierenkomponente für diese Szene verfügbar und sind bei zusammenarbeitenden Lernenden auch nötig.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Interaktive Lehrsitzungsszene und Bewertenkomponente
- Interaktive Lehrsitzungsszene und Verstehenkomponente
- Interaktive Lehrsitzungsszene und Rückmeldenkomponente
- Interaktive Lehrsitzungsszene und Helfenkomponente
- Interaktive Lehrsitzungsszene und Prozesssteuerungskomponente
- Interaktive Lehrsitzungsszene und Netzwerkkomponente
- Interaktive Lehrsitzungsszene und Kollaborierenkomponente

Temporärschicht Hilfeszene Schnittstellen zur Funktionalschicht

Die Hilfeszene wird dann aufgerufen, wenn in den Lehrsitzungsszenen aktiv oder passiv Hilfe angefordert wurde. Die Funktionen die passenden Inhalte für die aktuelle Situation zu bekommen, werden in der Helfenkomponente zur Verfügung gestellt. Da auch diese Komponente das Lehrwissen für den Nutzer an bestimmte Bedingungen anpasst, hilft hier die Funktionalität der Anpassenkomponente. Falls die Navigation innerhalb der Hilfe komplexer ist, kann zusätzlich die Bewertenkomponente eingesetzt werden um die Interaktionen im Kontext der Hilfe zu bewerten. Die Hilfe sollte allerdings keine Navigation erlauben, die vom Thema abdriftet. Damit die Szene den korrekten Ablauf einhält, wird die Prozesssteuerungskomponente nach den nächsten Schritten des Lehrablaufs gefragt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Hilfeszene und Anpassenkomponente
- Hilfeszene und Bewertenkomponente
- Hilfeszene und Helfenkomponente
- Hilfeszene und Prozesssteuerungskomponente

Temporärschicht Evaluationsszene Schnittstellen zur Funktionalschicht

Die Evaluationsszene benötigt nur sehr wenig Funktionalität. Sie muss die eingegebenen Informationen zur subjektiven Bewertung einer Lehrsitzung speichern. Die subjektiven Bewertungen des Lernenden zu der Lehrsitzung dienen zur besseren Einschätzung des Lernenden und zur Anpassung des Lehrmaterials hin zu seinen subjektiven Vorlieben, dass heißt nicht nur an seine real gemessenen Leistungen sind relevant, sondern auch an seine bewussten und unbewussten Einschätzungen, weil dies seine Motivation beeinflusst und damit indirekt den Wissenszuwachs. Eine Annäherung des Lehrmaterials an die

subjektiven Bedürfnisse erhöht die Motivation des Lernenden. Diese Evaluationen werden bewertet um sie abhängig von der Schwierigkeit der Lehrsitzung zur Fertigkeit des Lernenden besser einschätzen zu können. Die Schnittstelle zur Prozesssteuerungskomponente ist nötig, weil diese den nächsten Schritt bestimmt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Evaluationsszene und Bewertenkomponente
- Evaluationsszene und Prozesssteuerungskomponente

Temporärschicht Autorenszene Schnittstellen zur Funktionalschicht

Die Autorenszene verwaltet die Ansichten für die Autoren des Domänen- und Pädagogikwissens. Die benötigten speziellen Funktionen die nur für einen Autors genutzt werden erhält die Szene von der Autorenkomponente, darunter fallen Funktionen wie das Editieren des Domänen- und Pädagogikwissen, welche Auswirkungen eine Änderungen auf andere Wissenselemente hat, Statistiken wo noch Lehrmaterial fehlen könnte oder eine Versionierung und Verwaltung der Änderungen von mehreren Autoren. Die automatische Generierung von Annotierungen werden durch die Lehranreichernkomponente bereit gestellt, somit kann der Autor die Lehranreicherungsregeln an konkreten Lehrmaterial ausführen um sie direkt im Anschluss selbst zu überprüfen. Um Wissen zu vergleichen, auszutauschen oder neues Wissen von anderen Systemen zu integrieren gibt es zudem Zugriff auf die Zentralwissenkomponente. Um einen Wechsel von einer Szene zur nächsten zu ermöglichen, ohne das die Autorenszene die andere Szene kennt, wird eine Verbindung zur Prozesssteuerungskomponente benötigt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Autorenszene und Autorenkomponente
- Autorenszene und Lehranreichernkomponente
- Autorenszene und Zentralwissenkomponente
- Autorenszene und Prozesssteuerungskomponente

Temporärschicht Betreuerszene Schnittstellen zur Funktionalschicht

Die Betreuerszene stellt die Funktionen für den Betreuer bereit. Da dieser nur Aufsichtspersonen ist ohne inhaltlich das Lehrmaterial zu verstehen, sind die Anforderungen geringer. Er muss sowohl mit den Lernenden kommunizieren können als auch die Werkzeuge an die Hand bekommen um die Lernenden zu überwachen und nötigenfalls einzugreifen. Deshalb hat die Betreuerszene Zugriff auf die Funktionen der Autoritätenkomponente. Dies muss auch ohne dass dieser vor Ort ist möglich sein. Die Funktionen der Netzwerkkomponente sind unerlässlich, um die Kommunikation zwischen den jeweiligen Systemen von Lernenden und Betreuer zu verwalten. Um die Übergabe zu und von dieser Szene frei zu kontrollieren, wird die Prozesssteuerungskomponente benötigt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Betreuerszene und Netzwerkkomponente
- Betreuerszene und Autoritätenkomponente
- Betreuerszene und Prozesssteuerungskomponente

Temporärschicht Statistikszene Schnittstellen zur Funktionalschicht

Die Statistikszene will den Lernverlauf, den Lernfortschritt und eine Einschätzung des Lernenden vermitteln. Diese Funktionen sind bereits in dem Profil und dem gespeicherten Logbüchern größtenteils vorhanden. Deshalb ist es notwendig auf das gebündelte Lernendenwissen in der Zusammenfassungen und dem Profil zuzugreifen. Das Profil gibt Hinweise für generelle Tipps, bei denen die wichtigsten in die Statistikszene eingebunden werden können. Die Zusammenfassung dient der Übersicht für einen Menschen und zu eigenen Analysezwecken. Zudem wird diese Szene nur dann aufgerufen und geschlos-

sen, wenn der Prozess es vorsieht, dies wird in der Prozesssteuerungskomponente geregelt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Statistikszene und Zusammenfassenkomponente
- Statistikszene und Profilgenerierenkomponente
- Statistikszene und Prozesssteuerungskomponente

Temporärschicht Lehrerszene Schnittstellen zur Funktionalschicht

Die Lehrerszene muss es einer Lehrkraft ermöglichen durch Interaktionen seiner dazugehörigen Ansichten auszuhelfen. Eine Einsicht in den Interaktionsverlauf und begrenzte Kommunikation mit dem Lernenden, ohne am gleichen Ort wie der Lernende zu sein. Die Lehrerszene kann ihren Zweck erfüllen wenn eine Kommunikation von dem Lehrersystem zum Lernendensystem besteht. Da nur ein aktiver Nutzer pro System erlaubt ist, ist dies nur durch die Netzwerkkomponente möglich. Alle Funktionen die dem Lehrer erlaubt den Lernenden zu kontrollieren werden in der Autoritätenkomponente komplett abgedeckt. Um dem Lehrer eine eigene Einschätzung der Lernenden mit Daten außerhalb der aktuellen Lehrsitzung zu ermöglichen, werden die gut für die menschliche Analyse zusammengefassten Daten der Zusammenfassenkomponente verwendet. Die Prozesssteuerungskomponente ist für den reibungslosen Übergang von einer für die Lehrerszene unbekannten Szene verantwortlich und deshalb auch Teil dieser Szene.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Lehrerszene und Netzwerkkomponente
- Lehrerszene und Autoritätenkomponente
- Lehrerszene und Zusammenfassenkomponente
- Lehrerszene und Prozesssteuerungskomponente

Temporärschicht Programmverwaltungsszene Schnittstellen zur Funktionalschicht

Die Programmverwaltungsszene ist nicht im Lehrprozess aktiv, sondern dient rein zur Verwaltung der systemspezifischen Einstellungen und der Übersicht. Es steht über den angebotenen Features des ITS und bietet hauptsächlich das Hauptmenü und die Systemeinstellungen an. Die Hauptaufgaben dieser Szene werden in der Programmfunktionskomponente zur Verfügung gestellt, welche selbst auch nur für die Programmverwaltungsszene aufrufbar ist. Für die Synchronisierung und den Abgleich von Nutzerinformationen, wie zum Beispiel von Nutzername und Passwort können Netzwerkfunktionalitäten benötigt werden und damit auch die Netzwerkkomponente. Für Funktionen die mehrere Nutzer betreffen benötigt die Szene noch Zugriff auf die Mehrnutzerkomponente. Eine übergeordnete Funktion ist auch das Hoch- und Runterladen zu einer bestimmten (womöglich eine von vielen) Sammelstelle durch die Zentralwissenkomponente. Es ist damit grundsätzlich für jeden Nutzer technisch möglich dies zu unterbinden oder für seine Daten zu erlauben. Die Navigation wird nicht von der Programmverwaltungsszene übernommen, sondern von der Prozesssteuerungskomponente, da diese unabhängig von der Benutzeroberfläche ausgetauscht werden kann.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Programmverwaltungsszene und Netzwerkkomponente
- Programmverwaltungsszene und Zentralwissenkomponente
- Programmverwaltungsszene und Mehrnutzerkomponente
- Programmverwaltungsszene und Programmfunktionskomponente
- Programmverwaltungsszene und Prozesssteuerungskomponente

Funktionalschicht Profilgenerierenkomponente Schnittstellen zur Persistenzschicht

Die Profilgenerierenkomponente ist eines der zentralen Ankerpunkte des ITS um die Adaptivität zu gewährleisten. Die gesammelten Daten über den Nutzer führen zu einem Pro-

fil, deshalb greift diese Komponente auch auf alle Komponenten des Lernendenwissens zu. Dann ist es möglich aus den vergangenen Interaktionen mit dem informativen und interaktiven Lehrmaterial, die vergangenen Bewertungen der Eingaben in der Lehrsitzung und den abschließenden subjektiven Evaluationen eine Einschätzung generieren, welche die Fertigkeiten, Eigenarten, Vorlieben und Schwächen des Lernenden auf die ausschlaggebenden Faktoren reduziert. Für einen Verlauf der Profile und einen schnelleren Vergleich und eine Vermeidung einer Neuberechnung werden diese Profile auch in der Persistenzschicht gespeichert.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Profilgenerierenkomponente und Informatives Logbuch
- Profilgenerierenkomponente und Informative Bewertungen
- Profilgenerierenkomponente und Interaktives Logbuch
- Profilgenerierenkomponente und Interaktive Bewertungen
- Profilgenerierenkomponente und Evaluationen
- Profilgenerierenkomponente und Evaluationen Bewertungen
- Profilgenerierenkomponente und Zusammenfassungen
- Profilgenerierenkomponente und Profile

Funktionalschicht Vorschlagenkomponente Schnittstellen zur Persistenzschicht

Die Vorschlagenkomponente benötigt das aktuelle Profil von der Temporärschicht. Es schlägt zu dem Profil passendes Lehrmaterial vor, welches für eine Lehrsitzung verwendet werden kann. Dabei wird dieses gruppiert und unter Umständen wird erklärt warum es dies ausgewählt hat. Sodass eine Auswahl entsteht, die den Vorgaben entspricht und für verschiedene Anforderungen verwendet werden können. Zum Beispiel ist es möglich den Schwierigkeitsgrad, die Themen oder den Unterrichtsstil in verschiedenen Variationen anzubieten. Um so etwas umzusetzen benötigt es natürlich auch Zugang zu den Lehrmaterialien. Damit didaktisch Lehrmaterial passend zu dem Vorwissen des Lernenden ausgewählt wird, muss auch Zugriff auf die Lehrmaterialannotierungen gewährt werden.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Vorschlagenkomponente und Informatives Lehrmaterial
- Vorschlagenkomponente und Interaktives Lehrmaterial
- Vorschlagenkomponente und Informative Annotierungen
- Vorschlagenkomponente und Interaktive Annotierungen

Funktionalschicht Anpassenkomponente Schnittstellen zur Persistenzschicht

Die Anpassenkomponente benötigt Auswahlkriterien des Lehrmaterials und ein Profil von der Temporärschicht. Es benötigt nur ein einziges Profil und sollte deshalb nicht auf Informationen über den Lernenden aus der Lernendenwissenkomponente zugreifen. Damit lässt sich dann das Lehrmaterial so erstellen, dass es direkt in einer Lehrsitzung verwendet werden kann. Dazu sollten die Wissenselemente nach den gegebenen Kriterien aus den Informativen Lehrmaterial oder dem Interaktiven Lehrmaterial als Grundlage genommen werden. Es ist abhängig von dem Algorithmus der Anpassenkomponente welche Lehrmaterial wirklich verwendet wird, denn abhängig vom Profil wird das Lehrmaterial unterschiedlich stark reduziert werden. Um bessere Entscheidungen welches Lehrmaterial zur aktuellen Lehrsitzung passt und welche Teile davon verwendet werden sind die Metadaten unerlässlich

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Anpassenkomponente und Informatives Lehrmaterial
- Anpassenkomponente und Interaktives Lehrmaterial
- Anpassenkomponente und Informative Metadaten
- Anpassenkomponente und Interaktive Metadaten

Funktionalschicht Verstehenkomponente Schnittstellen zur Persistenzschicht

Die Verstehenkomponente soll Eingaben, die interpretiert werden müssen, in für das System eindeutige und handhabbare Informationen umwandeln. Darunter fällt zum Beispiel die Analyse von Freitexten in Schlüsselwörter. Für diese Aufgabe ist der Zugriff auf das Lehrmaterial erforderlich.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Verstehenkomponente und Informatives Lehrmaterial
- Verstehenkomponente und Interaktives Lehrmaterial

Funktionalschicht Bewertenkomponente Schnittstellen zur Persistenzschicht

Die Bewertenkomponente benötigt von der Temporärschicht eine Eingabe vom Lernenden, den Ablauf der Aktionen und das aktuelle Profil des Lernenden. Dann ist es ihr möglich die Eingabe zu bewerten. Im interaktiven Teil ist die Bewertung schwieriger, weil dort die Korrektheit der Eingabe im Sinne der Aufgabe ausgewertet wird und nicht statisch ist. Dazu muss die Bewertenkomponente das Domänenwissen ohne die Metainformationen kennen, um überhaupt den Inhalt, die Frage und/oder die richtige Antwort zu kennen. Also das Informative Lehrmaterial und das Interaktive Lehrmaterial. Um dies anwenden zu können, muss das Domänenwissen auch so aufgebaut sein, dass es für das System semantisch zuzuordnen ist. Dies bedeutet für das System eindeutiges und anwendbares Wissen oder zusätzlich korrekte und falsche Wege gespeichert zu haben. Darunter fallen Gleichungen, Auswertungsregeln, Algorithmen, formell korrekte Vorschriften, korrekte und falsche Lösungen und so weiter. Um einschätzen zu können wie schwierig die Aufgaben sind, werden die Annotierungen benötigt. Bei einer Fehlererkennung können die Missverständnisse verwendet werden um direkt häufige Fehler zu erkennen und zu kategorisieren, damit bei bekannten Fehlern direkt eine klare Bewertung getroffen werden kann. Das Ergebnis der Bewertung der Eingabe wird in den Bewertungen gespeichert um eine Historie der Bewertungen zu vervollständigen, welche später wiederverwendet werden können, zum Beispiel um den Lernenden einzuschätzen, den Verlauf der aktuellen Lehrsitzung einzuordnen oder um konkrete Vorkommen von Inhalten zu überprüfen. Da die Bewertungen als erstes direkt nach der Eingabe erfolgen und dies schon als als der beste Ort für die Speicherungsfunktionalität der Roheingaben erkannt wurde, braucht diese Komponente auch Zugriff auf die Logbücher und die Evaluationen.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Bewertenkomponente und Informatives Lehrmaterial
- Bewertenkomponente und Interaktives Lehrmaterial
- Bewertenkomponente und Informative Annotierungen
- Bewertenkomponente und Interaktive Annotierungen
- Bewertenkomponente und Missverständnisse
- Bewertenkomponente und Informatives Logbuch
- Bewertenkomponente und Informative Bewertungen
- Bewertenkomponente und Interaktives Logbuch
- Bewertenkomponente und Interaktive Bewertungen
- Bewertenkomponente und Evaluationen
- Bewertenkomponente und Evaluationen Bewertungen

Funktionalschicht Rückmeldenkomponente Schnittstellen zur Persistenzschicht

Die Rückmeldenkomponente benötigt von der Temporärschicht die bisherige Eingabe, die Auswertung dieser Eingabe und das aktuelle Profil. Die Rückmeldung ist eine für die Bewertung, die Eingabe des Lernenden und an den Lernenden angepasste Antwort. Es sollte sowohl eine Antwort generiert werden natürlich wirkt und nicht nur aus Stichpunkten besteht, als auch auf den Lernenden, dessen Eingabe und das Lehrmaterial eingehen. Für viele einfache Fälle können die Antworten ganz oder wenigstens größtenteils von Au-

toren vorgeschrieben werden, dies zählt vor allem für richtige Schritte oder bereits bekannte Probleme die in den Missverständnissen von der Bewertenkomponente gefunden wurden. Vordefinierte Rückmeldungen sind in den Rückmeldungen. Wenn man ein komplexeres Thema behandelt oder in einer Domäne bei dem die Eingaben unvorhersehbar sind, dann steigt die Menge an benötigten schon vorhandenen Antworten sehr schnell an. Dafür werden Regeln für das Generieren von Rückmeldungen benötigt um adaptiver auf die Geschehnisse zu antworten. Es wird eine zu den Eingabeparametern passende Rückmeldung durch eine Regel aus den Rückmeldungsregeln generiert.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Rückmeldenkomponente und Rückmeldungsregeln
- Rückmeldenkomponente und Rückmeldungen

Funktionalschicht Helfenkomponente Schnittstellen zur Persistenzschicht

Die Helfenkomponente benötigt die aktuelle Interaktionen und das aktuelles Profil von der Temporärschicht. Sie muss dem Lernenden helfen, wenn die gegebenen Informationen nicht ausreichen um sein Ziel zu erreichen. Die Helfenkomponente kann mehr Lehrmaterial passend zu der aktuellen Lehrsitzung, dem Interaktionsverlauf und dem Lernenden heraussuchen und anzeigen. Dies geht nur wenn der Helfenkomponente Zugriff auf die Lehrmaterialien selbst und den pädagogischen Inhalt davon hat. Die Annotierungen bilden zum Beispiel die Zusammenhänge der Lehrmaterialien untereinander ab. Um besser auf die Fehlinterpretationen der Lernenden eingehen zu können, helfen die Sammlung an Missverständnissen und was dann zu tun ist, damit wird fokussierter Hilfestellung gegeben.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Helfenkomponente und Informatives Lehrmaterial
- Helfenkomponente und Interaktives Lehrmaterial
- Helfenkomponente und Informative Annotierungen
- Helfenkomponente und Interaktive Annotierungen
- Helfenkomponente und Missverständnisse

Funktionalschicht Zusammenfassenkomponente Schnittstellen zur Persistenzschicht

Die Zusammenfassenkomponente hat die Aufgabe die Aktivitäten des Lernenden übersichtlicher zu komprimieren, sodass sie entweder durch das System selbst oder durch Menschen analysiert werden können um Schlüsse daraus zu ziehen. Die Anzahl der Ereignisse in Gruppen einzuordnen und diese Gruppen zu zählen wäre eine gute Aufgabe. Dafür muss die Komponente Zugriff auf die komplette Lernendenwissenkomponente geben, mit Ausnahme von Profile, weil dieses selbst auf die Zusammenfassenkomponente aufbaut. Der Zugriff auf die Zusammenfassungen wird gebraucht um die generierte Zusammenfassung langfristig zu speichern.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Zusammenfassenkomponente und Informatives Logbuch
- Zusammenfassenkomponente und Informative Bewertungen
- Zusammenfassenkomponente und Interaktives Logbuch
- Zusammenfassenkomponente und Interaktive Bewertungen
- Zusammenfassenkomponente und Evaluationen
- Zusammenfassenkomponente und Evaluationen Bewertungen
- Zusammenfassenkomponente und Zusammenfassungen.

Funktionalschicht Prozesssteuerungskomponente Schnittstellen zur Persistenzschicht

Die Prozesssteuerungskomponente besitzt keine Komponente in der Persistenzschicht zur Speicherung des Prozesses, weil der Prozess statisch ist und sich nicht ohne Austausch der Prozesssteuerungskomponente ändern soll. Die Funktionalität und der immer gleichbleibende Prozess einer konkreten Prozesssteuerungskomponente sind damit gewollt untrennbar. Sobald der Komponente die aktuelle Situation kennt, kann es den nächsten Schritt bestimmen. Es wird die Situation von der Temporärschicht erwartet und diese beinhalten schon alles was benötigt wird. Es sind die groben Informationen über den aktuellen Fortschritt des Lernens, entweder ist das direkt ein Element des Lehrprozesses oder Informationen die dazu führen.

Keine Verknüpfungen zu der Persistenzschicht erlaubt.

Funktionalschicht Autorenkomponente Schnittstellen zur Persistenzschicht

Die Autorenkomponente stellt alle Funktionen zur Verfügung, damit der Autor als Experte des Lehrmaterial und/oder der Lehre das Domänenwissen und das Pädagogikwissen bearbeiten kann. Die Funktionen vereinfachen die Auflistung und Editierung, darunter fällt die Filterung, Sortierung, Stapeleditierung, automatische Anpassungen, Anzeigen der Auswertungen, Vorschlagen von weiteren Inhalten oder Hinweise wo Bedarf besteht. Es muss also Zugriff von der Autorenkomponente auf die gesamte Domänenwissenkomponente und auf die gesamte Pädagogikwissenkomponente vorherrschen.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Autorenkomponente und Informatives Lehrmaterial
- Autorenkomponente und Informative Metainformationen
- Autorenkomponente und Interaktives Lehrmaterial
- Autorenkomponente und Interaktive Metainformationen
- Autorenkomponente und Lehranreicherungsregeln
- Autorenkomponente und Informative Annotierungen
- Autorenkomponente und Interaktive Annotierungen
- Autorenkomponente und Missverständnisse
- Autorenkomponente und Rückmeldungsregeln
- Autorenkomponente und Rückmeldungen

Funktionalschicht Lehranreichernkomponente Schnittstellen zur Persistenzschicht

Die Lehranreichernkomponente hat die Aufgabe aus allgemeinem didaktischem Wissen, den Lehranreicherungsregeln und konkret an das Domänenwissen gebundene Annotierungen zu erstellen. Dazu braucht es den Zugriff auf die Lehranreicherungsregeln an sich, welche dann ihrerseits das gesamte Domänenwissen abfragen können. Heraus kommen Informative und Interaktive Annotierungen die durch den jeweiligen Komponenten abgespeichert werden können.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Lehranreichernkomponente und Informatives Lehrmaterial
- Lehranreichernkomponente und Informative Metadaten
- Lehranreichernkomponente und Interaktives Lehrmaterial
- Lehranreichernkomponente und Interaktive Metadaten
- Lehranreichernkomponente und Lehranreicherungsregeln
- Lehranreichernkomponente und Informative Annotierungen
- Lehranreichernkomponente und Interaktive Annotierungen

Funktionalschicht Netzwerkkomponente Schnittstellen zur Persistenzschicht

Die Netzwerkkomponente regelt den Netzwerkverkehr von verschiedenen ITS-Ausprägungen zueinander. Damit bekannt ist, welche Aktionen der Lernende bereits getätigt hat,

ist ein Zugriff auf die Logbücher und die Evaluationen vorteilhaft. So kann eine Synchronisierung auch langfristig zurückreichen.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Netzwerkkomponente und Informatives Logbuch
- Netzwerkkomponente und Interaktives Logbuch
- Netzwerkkomponente und Evaluationen

Funktionalschicht Zentralwissenkomponente Schnittstellen zur Persistenzschicht

Die Zentralwissenkomponente basiert darauf das Wissen des ITS zu teilen und von anderer Quelle zu beziehen. Um zu einer Aussage über sehr vielen gesammelten Daten zuverlässig zu fällen, wird eine große Menge an Daten von heterogenen Lernenden benötigt, sei es das nur Teile oder das gesamte Wissen zur Verfügung gestellt wird. Deshalb benötigt die Zentralwissenkomponente Zugriff auf das gesamte Wissen. Die einzige Ausnahme dabei sind die Programmeinstellungen, weil diese konkret für genau dieses ITS-Programm bestimmt sind.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Zentralwissenkomponente und Informatives Lehrmaterial
- Zentralwissenkomponente und Informative Metadaten
- Zentralwissenkomponente und Interaktives Lehrmaterial
- Zentralwissenkomponente und Interaktive Metadaten
- Zentralwissenkomponente und Lehranreicherungsregeln
- Zentralwissenkomponente und Informative Annotierungen
- Zentralwissenkomponente und Interaktive Annotierungen
- Zentralwissenkomponente und Missverständnisse
- Zentralwissenkomponente und Rückmeldungsregeln
- Zentralwissenkomponente und Rückmeldungen
- Zentralwissenkomponente und Informatives Logbuch
- Zentralwissenkomponente und Informative Bewertungen
- Zentralwissenkomponente und Interaktives Logbuch
- Zentralwissenkomponente und Interaktive Bewertungen
- Zentralwissenkomponente und Evaluationen
- Zentralwissenkomponente und Evaluationen Bewertungen
- Zentralwissenkomponente und Zusammenfassungen
- Zentralwissenkomponente und Profile

Funktionalschicht Mehrnutzerkomponente Schnittstellen zur Persistenzschicht

Die Mehrnutzerkomponente als eine Komponente sorgt im Grunde genommen nur dafür, dass sich die Nutzerdaten nicht überschneiden. Zudem kann auch eine Authentifizierung stattfinden. Dazu muss bekannt sein welche Lernende es gibt und welche Einstellungen zu ihnen gehören. Sowohl die Einstellungen selbst als auch die Nutzerdaten werden in den Programmeinstellungen gespeichert. Jeder Nutzer hat seine eigenes Lernendenwissen. Die Mehrnutzerkomponente kann also das komplette Lernendenwissen einfach austauschen ohne wirklich Zugriff darauf zu haben (zum Beispiel kann sie diese nicht entschlüsseln aber für jeden Nutzer eine eigene verschlüsseltes Lernendenwissen bereithalten).

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Mehrnutzerkomponente und Programmeinstellungen

Funktionalschicht Kollaborierenkomponente Schnittstellen zur Persistenzschicht

Die Kollaborierenkomponente sorgt für eine Zusammenarbeit der Lernenden mit nur einem Lehrmaterial in einer gemeinsamen Lehrsitzung. Da jeder Nutzer eigene Aktionen tätigt, muss die Kollaborierenkomponente aufpassen, dass kein verzerrtes Lernendenwis-

sen erzeugt wird und die Bewertungen entsprechend der Hilfe der anderen beteiligten Lernenden nur die Eingaben bekommen. Somit wird der einzelne Lernende nicht überschätzt. Dazu braucht die Kollaborierenkomponente Zugriff auf die bisherigen Bewertungen. Da nur die aktuelle Lehrsitzung relevant ist, sind die Logbücher und Evaluationen aus vergangenen nicht notwendig. Die Aktionen der aktuellen Lehrsitzung sind in der Temporärschicht zugänglich.

Keine Verknüpfungen zu der Persistenzschicht erlaubt.

Funktionalschicht Autoritätenkomponente Schnittstellen zur Persistenzschicht

Die Autoritätenkomponente gibt einer Person die Berechtigung einen Lernenden zu überwachen und mit ihm zu kommunizieren. Die Autoritätenkomponente braucht keinen Zugriff auf das Domänenwissen. Es ist nicht Aufgabe der Autoritäten den Unterricht zu gestalten, denn dem Lernenden wird bereits all das zum Lernen benötigte Wissen bereit gestellt. Die Temporärschicht übergibt den aktuellen Lehrrahmen. Es ist nicht die Aufgabe einer Autorität neues Lehrmaterial zu weiteren Lehrmaterialien zu verlinken, sondern von der Helfenkomponente. Die Autoritätenkomponente kann allerdings nur auf die aktuellen Interaktionen des Lernenden zugreifen, welche in der Temporärschicht gespeichert sind und übergeben werden. Um grob eine Übersicht des Lernenden zu haben hat die Komponente Zugriff auf die Zusammenfassungen und Profile. Auch wenn keine Lehrempfehlungen in einem Thema gemacht werden können, so gibt ein grober Einblick in die Historie des Lernenden den Autoritäten Hinweise, wie mit ihm umgegangen werden sollte. Zudem kann es bei der Kommunikation zwischen Nutzern sinnvoll sein, die Programmeinstellungen einzusehen und für den Lernenden anzupassen. Dies trifft insbesondere für den Betreuer des ITS zu, weil diese sich mit dem ITS auskennen müssten.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Autoritätenkomponente und Zusammenfassungen
- Autoritätenkomponente und Profile
- Autoritätenkomponente und Programmeinstellungen

Funktionalschicht Programmfunktionskomponente

Es werden die Einstellungen gespeichert, die für das System relevant sind und über einen Neustart hinaus zugänglich sein müssen, darunter fallen die Schriftgröße, Farbthema der Benutzeroberfläche, ob an die Nutzung erinnert werden soll und so weiter. Auch wichtig sind die Nutzerdaten die nicht das Lernen betreffen wie dem Nutzernamen, Passwort oder andere Daten über den Nutzer die zu Forschungszwecken relevant sein können. Damit können mehr als ein Nutzer verwaltet werden. Zudem können Informationen, die nicht für Nutzer direkt für die Funktionen relevant sind, verwaltet werden, wie den Kontakt zu den Entwicklern, rechtliche Hinweise und so weiter. Da all diese Daten in der Persistenzschicht in der Komponente der „Programmeinstellungen“ abgebildet ist, wird nur eine Schnittstelle dahin benötigt.

Erlaubte Verknüpfungen sind folgend aufgezählt.

- Programmfunktionskomponente und Programmeinstellungen

III.2.i. Softwarearchitektur und ITS-Lehrprozess gemeinsam

Es wurde in der Architektur nicht nur die Funktionalität in einzelne Funktionskomponenten aufgeteilt, sondern auch darauf geachtet den ITS-Lehrprozess leicht damit verwenden zu können. Die Anwendung des Prozesses mit der Architektur ist damit vereinfacht und die Verwendung beider erhöht die spezifizierten Bereiche des Gesamtsystems. Betrachtet man die Softwarearchitektur und den Lehrprozess gemeinsam so erkennt man Reihenfolgen an Komponenten die aufgerufen werden sollten um effektiv zu sein. Es ist nicht nur strukturell, sondern auch zeitlich nur eine begrenzte Kombination von Aktionen mit den

einzelnen Komponenten in einer Reihenfolge erlaubt. Eine Komponente darf an bestimmten Stellen in der Reihenfolge von Aktionen verwendet werden. Die Funktion der Komponenten im Ganzen zu sehen ist dabei wichtig: Zum Beispiel die Bewertenkomponente dann als Input und die Zusammenfassen-/Profilgenerierenkomponente als ein Output zum Erfolg des Systems zu sehen. Die Reihenfolgen der Aktionen richten sich allgemein an dem ITS-Lehrprozess aus, werden aber von der Softwarearchitektur und den jeweiligen Systemanforderungen spezieller. Der Lehrprozess wird damit spezialisiert, sowohl in Use-Case-Prozessen als auch auf technische Sicht in der Reihenfolge der Komponenten die Verwendet werden. Wenn man konkret wird, dann nimmt man auf der technischen Sicht die Namen der Klassen, Objekte, Methoden und so weiter für jeden einzelnen Schritt. Das verändert nicht die Reihenfolge.

Um den Lehrprozess zu spezialisieren ohne konkret zu werden und abstrahiert genug zu sein um auf mehreren unterschiedlichen Systemen verwendet zu werden, kann dies naiv angegangen werden. Man hat die Stütze des Lehrprozesses und kann geplante häufig vorkommende Abläufe in das Schichtmodell eingetragen. Einige häufig vorkommende Abläufe werden auch für die Entwicklung des Prototypen modelliert. Hier als Beispiel der Ablauf, dass der Lernende Lehrsitzungen vorgeschlagen bekommt und sich davon eine aussucht. Selbst mit einer farblichen Annotierung wird eine visuelle Darstellung sehr schnell unübersichtlich, deshalb hier nur textuell. Betrachten wird ein idealtypischer Fall, bei dem sich ein einzelner Lernenden mit nur einem lokalen ITS ein Thema wählt, dann könnte dies die folgende Abfolge von Aktivitäten in den Komponenten mit den vorhandenen Komponenten geben.

Beispielabfolge der Komponenten eines einzelnen Lernenden bei der Themenwahlszene:

1. Eine konkrete Ansicht der Interaktionsschicht, wie zum Beispiel dem Hauptmenü, wird von der Programmverwaltungsszene unterstützt. Der Beginn des Lehrprozesses und damit die Übergabe an die Themenwahlszene wird zum Beispiel durch ein drücken eines Knopfes auf der konkreten Ansicht initiiert
2. Themenwahlszene übernimmt und zeigt neue konkrete Ansicht an. Es verwaltet Informationen über die Vorbereitung der Lehrsitzung. Es erfragt den nächsten Schritt bei der Prozesssteuerungskomponente
3. Prozesssteuerungskomponente entscheidet über den nächsten Schritt, in diesem Fall die Generierung eines Profils aus vorhandenen Daten des Logbuchs über den Lernenden und übergibt die Anweisung an die Themenwahlszene
4. Themenwahlszene spricht die Zusammenfassenkomponente an um eine Zusammenfassung zu generieren
5. Zusammenfassenkomponente benötigt Zugriff auf vorhandene Daten vom Lernenden, Anfrage beim Informativen Logbuch
6. Informatives Logbuch sucht die benötigten Daten heraus und übergibt diese
7. Zusammenfassenkomponente braucht auch interaktive Daten
8. Interaktives Logbuch sucht heraus und übergibt benötigte Daten
9. Zusammenfassenkomponente braucht auch Informative Bewertungen der Eingaben vom Lernenden
10. Informative Bewertungen sucht heraus und übergibt benötigte Daten
11. Zusammenfassenkomponente braucht auch Interaktive Bewertungen der Eingaben vom Lernenden
12. Interaktive Bewertungen sucht heraus und übergibt benötigte Daten
13. Zusammenfassenkomponente braucht auch subjektive Einschätzungen der Lehrsitzungen
14. Evaluationen sucht heraus und übergibt benötigte Daten
15. Zusammenfassenkomponente braucht auch Evaluationen Bewertungen der Eingaben vom Lernenden

16. Evaluationen Bewertungen sucht heraus und übergibt benötigte Daten
17. Zusammenfassenkomponente generiert Profil anhand gesammelter Daten
18. Zusammenfassungen speichert neue Zusammenfassungen und bestätigt dies
19. Zusammenfassenkomponente übergibt das neu erstellte Zusammenfassung zurück an die Themenwahlszene
20. Themenwahlszene spricht die Profilgenerierenkomponente an
21. Profilgenerierenkomponente benötigt Zugriff auf vorhandene Daten vom Lernenden
22. Informatives Logbuch sucht die benötigten Daten heraus und übergibt diese
23. Profilgenerierenkomponente braucht auch interaktive Daten
24. Interaktives Logbuch sucht heraus und übergibt benötigte Daten
25. Profilgenerierenkomponente braucht auch Informative Bewertungen der Eingaben vom Lernenden
26. Informative Bewertungen sucht heraus und übergibt benötigte Daten
27. Profilgenerierenkomponente braucht auch Interaktives Bewertungen der Eingaben vom Lernenden
28. Interaktives Bewertungen sucht heraus und übergibt benötigte Daten
29. Profilgenerierenkomponente braucht auch subjektive Einschätzungen der Lehrsitzungen
30. Evaluationen sucht heraus und übergibt benötigte Daten
31. Profilgenerierenkomponente benötigt die Evaluationen Bewertungen
32. Evaluationen Bewertungen sucht heraus und übergibt benötigte Daten
33. Profilgenerierenkomponente benötigt die letzte Zusammenfassung
34. Zusammenfassungen sucht heraus und übergibt benötigten Daten
35. Profilgenerierenkomponente generiert Profil anhand gesammelter Daten
36. Profile speichert neues Profil und bestätigt dies
37. Profilgenerierenkomponente übergibt das neu erstellte Profil zurück an die Themenwahlszene
38. Themenwahlszene hat die Aufgabe erfüllt und behält für weitere Aufgaben das aktuelle Profil bei. Es erfragt nächste Lehrprozessanweisungen
39. Prozesssteuerungskomponente bestimmt dem Lernenden die Themen vorzuschlagen und übergibt die Anweisungen
40. Themenwahlszene benötigt Funktionalität für das Vorschlagen von Themen
41. Vorschlagenkomponente bekommt das aktuelle Profil vom der Temporärschicht (vermeidet Bindung an die Profilgenerierenkomponente und an die Profile) und benötigt zusätzlich Informatives Lehrmaterial
42. Informatives Lehrmaterial sucht benötigte Daten heraus und gibt diese zurück
43. Vorschlagenkomponente benötigt auch die Annotierungen für das informative Lehrmaterial
44. Informative Annotierungen sucht benötigte Daten heraus und übergibt diese zurück
45. Vorschlagenkomponente hat passendes Domänenwissen in vorzuschlagenden Themen gruppiert und übergibt die Gruppen mit zusammenfassenden Informationen und Kriterien zum finden des Domänenwissens an die Themenwahlszene zurück
46. Themenwahlszene initiiert eine konkrete Ansicht für die Anzeige der groben Informationen über die Themen mit der Möglichkeit eines zu wählen
47. Nach Wahl eines Themas durch den Lernenden fragt die Themenwahlszene nächsten Schritt bei der Prozesssteuerungskomponente
48. Prozesssteuerungskomponente bestimmt, dass die Wissenselemente passend zum gewählten Thema herausgesucht werden müssen und gibt dies zurück
49. Themenwahlszene benötigt generiertes informatives und interaktives Lehrmaterial

50. Die Anpassenkomponente bekommt die Kriterien für die Auswahl der Wissenselemente, sie benötigt die konkreten Wissenselemente
51. Informatives Lehrmaterial gibt Wissenselemente zurück die den Vorgaben entsprechen
52. Die Anpassenkomponente benötigt auch passend zu dem informativen Lehrmaterial das interaktive Lehrmaterial
53. Interaktives Lehrmaterial gibt Wissenselemente zurück die den Vorgaben entsprechen
54. Die Anpassenkomponente sequenzialisiert, reduziert und wandelt die Wissenseinheiten in an den Lernenden angepasste Lehrmaterial um, welches als Grundlage für eine Lehrsitzung verwendet werden kann und gibt dies der Themenwahlszene zurück
55. Die Themenwahlszene hat jetzt an den Lernenden angepasstes Lehrmaterial und fragt nächste Schritte bei der Prozesssteuerungskomponente an
56. Prozesssteuerungskomponente bestimmt, dass die Lehrsitzung beginnen kann und sucht die nächste Szene raus: Informative Lehrsitzungsszene
57. Themenwahlszene erlaubt es die Lehrsitzung zu starten, dies wird der Interaktionsschicht mitgeteilt, die zum Beispiel durch einen Knopf in der konkreten Ansicht realisiert wird
58. Bei Rückmeldung an die Themenwahlszene, dass die Lehrsitzung beginnen kann, wird an die Informative Lehrsitzungsszene übergeben. Das an den Lernenden angepasste Lehrmaterial und andere wichtige Daten in der Temporärschicht, wie das aktuelle Profil, die noch benötigt werden bleiben erhalten

Die Abläufe werden sehr lang, auch wenn, wie hier, nur die abstrahierten Komponenten betrachtet werden. Zusätzlich gibt es eine sehr große Anzahl an korrekten Abläufen. Sie aufzuzählen hilft so nicht weiter, stattdessen wird abstrahiert um die Muster zu erkennen, die in diesen Abläufen vorkommen. Diese Abläufe fangen vom Nutzer aus an und das bedeutet aus der Interaktionsschicht. Da dort aber keine Komponenten definiert sind und unter Umständen diese viel zu kleinteilig und konkret sind, kann mit den Komponenten der Temporärschicht als Ausgangspunkt in einem Ablauf begonnen werden. Eine Komponente der Temporärschicht kann nun Funktionen der Funktionalschicht aufrufen, diese wiederum können abstrahierte Versionen der Datenbank durch die Persistenzschicht aufrufen. Die Persistenzschicht kümmert sich dann um die Speicherung indem sie die konkreten Datenbanken in der Datenerhaltungsschicht anspricht. Die

Ein Nutzer wird natürlich nicht nur in einer Szene bleiben, jede Szene für sich benötigt für den reibungslosen Ablauf einen temporären Zustand mit bestimmten Variablen. Einige diese Variablen können aber nicht von der Szene selbst gesetzt werden. Eine Evaluationsszene aufzurufen ohne jeglichen Kontext ergibt keinen Sinn. Wenn nach dem vordefinierten Prozess vorgegangen wird, dann ergeben die Abläufe automatisch den Wechsel zwischen den Szenen. Eine Szene die durch eine andere ersetzt wird kann also die benötigten Informationen bei der Transition übergeben. Sodass zum Beispiel die Themenwahlszene am Ende die Informative Lehrsitzungsszene aufruft und das angepasste zu präsentierende Lehrmaterial beim Wechsel der Szene übergibt. Die Struktur des temporären Zustands kann sich also beim Szenenwechsel in seinem Inhalt ändern und noch verlangte Argumente von der vorherigen Szene in diesem Zustand gespeichert werden.

Die Anzahl der konkreten Abfolgen sind sehr viele, es reichen aber Ausschnitte für die Möglichkeiten die dem System als nächsten Schritt offen stehen. Mit den Abfolgen lassen sich die Aufrufe der unterschiedlichen Komponenten untereinander schlecht darstellen. Die Möglichkeiten zu kennen und zu beschränken ist aber Grundlage für eine abstraktere Beschreibung. Denn letztendlich sind es die Abläufe und darunter hauptsächlich der ITS-Lehrprozess, welche die Funktionalitäten die als nächstes benötigt werden festlegen. Da die Funktionalitäten schon in Schichten und Komponenten eingeteilt wurden, ist es nun

möglich die nächsten Komponente zu benennen, die benötigt wird. Es werden alle sinnvollen Abläufe überprüft, sodass damit alle notwendigen Aufrufe zwischen Komponenten beschrieben werden können. Alle notwendigen Aufrufe brauchen Schnittstellen.

Ganz im Sinne des Softwareengineering ist es also zu empfehlen sich eine Softwarearchitektur und einen Prozess für eine ITS-Implementierung vorher zu überlegen. Denn viele Fehler lassen sich in einer Ad-Hoc-Entwicklung nicht vorher erkennen und der Überblick kann schneller verloren gehen. Die vorgestellte Architektur und der Prozess bieten einen hervorragenden Leitfaden in der Entwicklung eines ITS.

IV. IMPLEMENTIERUNG

IV.1. Implementierung des Softwareframeworks

Um das in dem vorherigen Kapitel vorgestellte Konzept auch umzusetzen, wird nicht direkt ein ITS programmiert, sondern zur erhöhten Wiederverwendbarkeit und zur einfachen Umsetzung der Spezifikation des Konzepts und der beinhalteten Softwarearchitektur wird zuerst ein Softwareframework entwickelt. Dessen Aufgabe ist es ein Konstrukt zur Verfügung zu stellen, welches die Komponenten und Restriktionen schon integriert hat, sodass entlang dieses Rahmens implementiert werden kann. Es ist eine mögliche Implementation die zeigen soll, dass es möglich ist, diese Architektur in der vorgestellten Art und Weise auch so programmatisch umzusetzen. Es ist wünschenswert, wenn es mehrere unterschiedliche Softwareframeworks in verschiedenen Implementierungen gibt.

IV.1.a. Wahl der Programmiersprache

Für die Implementierung muss eine Programmiersprache gewählt werden. Diese dient als Grundlage für die Implementierung. Um es nicht unnötig kompliziert zu machen, wird nur eine Programmiersprache gewählt. Das Softwareframework kann abhängig von der Programmiersprache dann nur noch von bestimmten anderen Programmiersprachen verwendet werden. Als Programmiersprache für das Softwareframework wird Swift gewählt. Dies hat mehrere Gründe die folgend aufgezählt sind, siehe auch Tabelle R-IV1a-1.

- Swift ist neu, sie hat wichtigen Aspekte der Programmierung modern umgesetzt. Eine ältere Sprache hat Fehleranfälligere Konzepte und ist dadurch auch unattraktiv für einige Neuprojekte.
- In Swift lassen sich viele der benötigten generische Konstrukte direkt umsetzen, damit können viele Spezifikationen des Konzepts auch in der Programmiersprache ähnlich abgebildet werden.
- Swift wird von einer Firma entwickelt, die immer wieder an der Börse als die reichste Firma der Welt gehandelt wird: Apple. Das gibt Vertrauen für eine langfristige Wartung eben jener.
- Swift ist Open Source und wird durch einen offenen Prozess mit der Gemeinschaft weiterentwickelt.
- Plattformübergreifend lässt sich Swift auf macOS, Linux Ubuntu und Microsoft Windows ausführen.
- Swift als kompilierte Sprache mit starker Typisierung - Typen sind zur Kompilierung bekannt - ist sehr performant, im Durchschnitt performanter als interpretierte Sprachen wie Python, PHP oder JavaScript
- Die Lernkurve und Fallstricke sind in Swift geringer als in C++ oder C, zum Beispiel sind Adresszeiger fast nie notwendig in Swift. Es ist ähnlich intuitiv wie andere neuere Sprachen wie Python3, Kotlin oder Rust
- Swift ist sehr beliebt in verschiedenen Rankings. Beim Ranking durch Suchmaschinen [2017 TIOBE.COM] Platz 12 und bei einer GoogleTrends Analyse [2018 PYPL POLULARITY OF PROGRAMMING LANGUAGE] ist Swift bei Platz 9
- Swift wird stark verwendet in StackOverflow und GitHub [2018 REDMONK], den Ressourcen für Programmierer. Und ist trotz seines geringen Alters schon in den Top10.

Der nachvollziehbare Nachteil von Swift ist, dass einige andere Softwareframeworks, die in einem ITS-Entwicklungsprojekt nützlich sein könnten, nicht auf allen Plattformen vorhanden sind. Die meisten Programme von Swift sind bisher auf Plattformen auf Basis von Darwin oder auf Ubuntu. Swift und C# fokussieren mehr eine Platform, welches eine sehr gute Einbindung in die Betriebssysteme führt, weil die dazugehörigen Bibliotheken und Frameworks zusammen mit den Programmiersprachen weiterentwickelt werden. Bei Swift wurde die Schwäche erkannt und es wird aktuell aktiv an eine Stärkere Einbindung in

Windows gearbeitet. Die Programmiersprache Kotlin für Android Google sind hingegen plattformabhängig und es stehen keine Pläne an dies zu ändern.

C# ist mit der notwendigen .NET-Bibliothek auf Microsoft Windows verfügbar. Eines der Hauptszenarien, welches weiter oben in der Doktorarbeit beschrieben worden ist, benötigt mobile Endgeräte. Programmieren als ein Vorreiter für die technologische Entwicklung von Geräten in der Lehre sollte schon auf mobilen Endgeräten stattfinden und nicht PCs für die Programmierung verwenden [2012 TILMANN, MOSKAL and DE HALLEUX]. Für einen Pre-Standard, dessen Anwendung in zukünftig entwickelnden ITSs liegt, muss nicht nur die Programmiersprache diesen Anforderungen gerecht werden, sondern auch die Plattformen auf denen sie betrieben wird. Im Hinblick, dass Microsoft das Betriebssystem Windows Mobile eingestellt hat, sind PCs zwar landläufig vorhanden aber nicht Mittel der Wahl für die aufkommenden ITSs in den nächsten zehn Jahren.

Kotlin von Google stellt eine Alternative als gleichwertige Sprache zu Java in der Android Programmierung. Kotlin ist als Programmiersprache de facto nur auf Android zu finden und damit in der Verbreitung der Plattformen auf mobile Geräte beschränkt. Eine zu starke Beschränkung, auch wenn die unterstützten Sprachkonstrukte und Sicherheit fast Swift-Niveau erreicht, so wird die Weiterentwicklung der Sprache durch den Zwang, das der entstandene Bytecode in der Java Virtual Machine (JVM) laufen muss, beschränkt. Plattformübergreifende beliebte Sprachen sind C, C++, JavaScript, PHP, Python und Java.

C ist eine Programmiersprache die sehr hardwarenah ist, weil man auf einem geringem Abstraktionsniveau stark optimieren kann und muss. Zum Beispiel muss man sich häufig bytegenau über die Deklarationen Gedanken machen. Sie ist sehr alt und besitzt wenig Sicherheitsmechanismen und es kann nicht objektorientiert programmiert werden. Die Zeigerarithmetik mit Arrays sind da ein gutes Beispiel: Ein Array ist nur ein Zeiger auf das erste Element des Arrays, das letzte Element muss NULL sein. Der Index des Arrays ist ein Integer und dieser kann eine beliebige Zahl sein, welcher dann einfach auf willkürliche Speicherbereiche außerhalb des Arrays zugreifen kann, wenn dieser falsch gewählt wird. Für ein derart schwieriges Projekt wie ein ITS ist C als die Programmiersprache in der hauptsächlich programmiert wird sehr ungeeignet.

C++ ist eine sehr breit eingesetzte Sprache. Allerdings hat sie einige Nachteile. Sie ist eine Programmiersprache die weniger Wert auf Sicherheit legt, dies mag unter Umständen an ihrem Alter und den performanzkritischen Einsatzzweck in hardwarenahen Bereichen liegen. Zu den unsicheren Funktionen zählt zum Beispiel die Zeigerarithmetik, Mehrfachvererbung, undefinierte Variablen und keine Ausnahmen bei falschem Index in Listen. Die Laufzeitumgebung besitzt auch keinen Garbage-Collector. Diese Programmiersprache erlaubt zu viele Fehler für einen Gewinn an Performanz der in grafisch aufwändigen Anwendungen entscheidend sein kann und nur in absoluten Ausnahmefällen auch in sehr speziellen Teilen eines ITS notwendig wäre. Solch einen Teil eines ITS könnte man einfach in einer noch performanteren und globaleren Sprache wie C schreiben. Das Gros sollte aber nicht davon bestimmt werden.

Die Performanz ist bei den restlichen Sprachen (JavaScript, PHP, Python und Java) im Allgemeinen schlechter als bei Programmiersprachen, die keinen Interpreter benötigen. Java stellt hierbei eine Mischung dar, weil erst zu Bytecode kompiliert wird, dieser dann aber von der JVM interpretiert werden muss, um vom Prozessor ausgeführt zu werden. JavaScript, PHP und Python sind Skriptsprachen, die nicht dafür entwickelt wurden für sehr große komplexe Programme eingesetzt zu werden. JavaScript und Python sind ohne starke Typisierung eher unpassend für sehr komplexe Programme wie ITS-Entwicklungen. ITSs sind von Natur aus sehr schwierig und die Quellcodebasis wird wahrscheinlich sehr groß. Es ist allgemein schwieriger fehlerfreien Code in einem komplexen Projekt mit viel Code ohne statische Typisierung zu programmieren. JavaScript und PHP sind beide fast ausschließlich für die Webentwicklung im Einsatz. JavaScript ist bis auf sehr wenige Aus-

nahmen vollständig abwärtskompatibel und damit geht die starke Inkonsistenz einher. Javas Beliebtheit und Installationsbasis ist in den letzten Jahren stark gesunken, dass liegt auch daran, dass immer eine virtuelle Machine benötigt wird, die den Zwischencode „Bytecode“ interpretieren kann. Viele Sicherheitslücken, der Einsatz von Kotlin als gleichwertige Programmiersprache in Android und die verringerte Notwendigkeit von Java-Applets durch die Integration mehr Funktionen in HTML5/JavaScript 6 führen zu einer ungewisseren Zukunft von Java.

Alles in Allem stehen mehrere Programmiersprachen zur Auswahl für die Entwicklung eines ITS mit ihren jeweiligen Vor- und Nachteilen. Ein Softwareframework sollte dementsprechend auch in der Sprache programmiert worden sein, in den das ITS programmiert wird, denn die Interoperabilität zwischen den einzelnen Programmiersprachen ist selten gut. Deshalb ist eine Softwareframework, dass für alle Programmiersprachen gleichermaßen gut ausgelegt ist unpraktikabel. Es ist anzuraten für bestimmte Programmiersprachen jeweils ein eigenes Framework zu erstellen. Solch ein Vorgehen ist auch für andere Konzepte in der Programmierung üblich. Wie im Beispiel des „Reactive Programming“ werden Softwareframeworks für die unterschiedlichsten Programmiersprachen entwickelt mit jeweils dem Präfix „Rx“ als Erkennungsmerkmal: Rx.NET für C#/Visual Basic, RxCpp für C++, RxJS für JavaScript, RxPHP für PHP, Rx.rb für Ruby, Rx.py für Python, RxJava für Java, RxSwift für Swift, RxScala für Scala, RxKotlin für Kotlin und so weiter. Empfehlenswert für ITS sind Swift und C# für eine plattformabhängige Sprache und JavaScript für eine webbasierte Implementierung.

Swift	Rust	Java	JavaScript	C++	Python	C#	C
+ Moderne Sprachkonzepte	+ Moderne Sprachkonzepte	+ Stark verbreitet	+ Quasi immer installiert	+ Kompilierte Sprache	+ Verbreitet in der KI-Entwicklung	+ Zukunftssicher durch Microsoft (Hauptprogrammiersprache)	+ Stark verbreitet
+ Statische Typen (Zuverlässigkeit und Sicherheit)	- Einsatz vor allem in der Serverentwicklung	+ Sowohl für mobile Geräte als auch Desktoprechner geeignet	+ Sowohl für mobile Geräte als auch Desktoprechner geeignet	+ Statische Typen (Zuverlässigkeit und Sicherheit)	- Dynamische Typisierung in komplexen Projekten fehleranfälliger	+ Statische Typen (Zuverlässigkeit und Sicherheit)	+ Sehr performant
+ Kompilierte Sprache (Performanz	- Wenig verbreitet	- Wenig moderne Sprachkonzepte	- Hauptsächlich in der Webentwicklung	+ Sehr verbreitet	- Interpretierte Sprache (schlechtere Performanz)	- Hauptsächlich Einsatz in Windows	- Alte Sprachkonzepte
+ Zukunftssicher durch Apple (Hauptprogrammiersprache)	- Umsetzung auf mobile Endgeräten nicht geplant	- Benötigt im Hintergrund laufende JVM	- Sehr uneinheitlich durch vollständige Abwärtskompatibilität	- Umsetzung auf mobile Endgeräten nicht geplant	- Umsetzung auf mobile Endgeräten nicht geplant	- Umsetzung auf mobile Endgeräten nicht geplant	- Umsetzung auf mobile Endgeräten nicht geplant
+ Sowohl für mobile Geräte als auch Desktoprechner geeignet			- Interpretierte Sprache (schlechtere Performanz)	- Alte Sprachkonzepte			- Hauptsächlich Systemprogrammierung
- Wenig verbreitet auf Windows							

Tabelle R-IV1a-1: Tabelle der Vor- / Nachteile möglicher Implementierungssprachen

IV.1.b. Ziele des Softwareframeworks

Es soll ein generisches Softwareframework erzeugt werden, generisch deshalb, weil das Softwareframework ohne gegebene Typen nicht funktioniert, entweder vorgefertigt oder vom Nutzer des Softwareframeworks implementiert. Das Softwareframework wird nicht komplett implementierte Komponenten anbieten, sondern Komponenten welche Vorgaben machen aber noch Teile besitzen die konkretisiert werden müssen. Die konkrete Ausführung von der Komponente ist die zusätzliche Arbeit, welche erbracht werden muss, um eine funktionsfähige Komponente für ein ITS zu haben. Es ermöglicht eine Implementie-

rung die an die Domäne angepasst ist und dabei automatisch die Architekturvorgaben erfüllt um damit viel Systementwicklung abnimmt.

Die allgemeinen Ziele des Softwareframeworks sind Beschreibungen, welche Eigenschaften das Softwareframework haben sollte, um möglichst nützlich für deren Nutzer zu sein. Diese werden folgend aufgezählt.

- Leichter Austausch der Komponenten durch den modularen Aufbau
- Restriktive Kommunikationsprotokolle zwischen den Komponenten. Das heißt eine möglichst geringe Anzahl an Verknüpfung einer Komponente zu anderen Komponenten. Wenn möglich geringer als in der Architektur vorgegeben.
- Wenn möglich sollten die konkretesten Spezifikationen genommen werden um nah an der Architektur zu bleiben

Ziel ist eine Einbindung mittels einer FRAMEWORK-Datei zu einem Xcode-Projekt so einfach wie möglich zu gestalten.

Nicht zu verwechseln ist ein Softwareframework mit einer Library. Eine Library stellt eine Vielzahl von Funktionen und Klassen bereit, welche in die Struktur und den Ablauf des eigenen Programms eingefügt werden kann, während ein Softwareframework das Prinzip des Inversion of Control (IOC) besitzt, welches nicht nur einzelne kleinere Komponenten bereitstellt, sondern eine Struktur und einen Ablauf schon vorgefertigt hat. Ein ITS-Softwareframework hat den Vorteil, dass der Aufwand zum erzeugen eines ITS stark verringert wird. Häufig leidet darunter die Anpassungsfähigkeit an bestimmte Domänen mit ihren teilweise einzigartigen Anforderungen.

Die Umsetzung der Architektur sollte drei grobe Designprinzipien erfüllen die folgend aufgezählt sind.

- Klare Verantwortlichkeiten definieren
- Vereinfachung durch Einschränkung ermöglichen
- Möglichen Informationsfluss bestimmen

Die Verantwortlichkeiten der einzelnen Komponenten wurden in dieser Doktorarbeit weiter oben beschrieben und nur dieser Funktionsumfang sollte von der jeweiligen Komponente abgedeckt werden. Es ist bei einer Implementierung ratsam, nicht nur genau diese Komponenten als Klassen umzusetzen, sondern sie als eine verantwortliche Instanz zu sehen. Dessen Aufgabe ist es, als Schnittstelle für den gesamten, für die jeweilige Komponente definierten, Funktionsumfang zu dienen. Ob sich die Implementierung dieser Funktionen dann innerhalb dieser Klasse befindet oder sie in noch speziellere Klassen delegiert wird, ist nicht relevant für die Einhaltung der Architektur, für das Softwaredesign des konkreten ITS aber schon. Wichtig ist nicht nur, dass die Komponenten ihre eigenen Verantwortlichkeiten erfüllen, sondern auch, dass sie ihre Verantwortlichkeiten nicht überschreiten und dann Aufgaben von anderen Komponenten übernehmen. Eine Überschreitung der Verantwortlichkeiten kann zu einem Konflikt zwischen Komponenten führen, Überfrachtung der Komponente selbst mit zu vielen Aufgaben erzeugen oder einer unklaren Struktur und Aufbau größerer Teile des ITS mit einer damit einhergehenden verschlechterten Wartung und Weiterentwicklung.

Das zweite Prinzip, dem gefolgt wird, ist die Verringerung der Komplexität durch die Einschränkung bestimmter Möglichkeiten der Veränderungen. Die Architektur selbst definiert keine Möglichkeit der Veränderung einer Komponente oder der Schnittstellen untereinander, doch ist es bei Architekturen nur bedingt möglich einzuschränken. Durchgesetzt werden kann dies bei der Umsetzung in einem Framework. Die Umsetzung der Architektur erfordert, dass allgemein beschriebene Komponenten konkretisiert werden, indem unter anderem Attribute und Methoden definiert werden, um die funktionalen Anforderungen zu erfüllen. Diese werden dann reduziert und optimiert um Performanzgrenzen einzuhalten für die nichtfunktionalen Anforderungen. Um die Nutzung des Frameworks und die Wartung zu verbessern wird die Sichtbarkeit, Kommunikationsfähigkeit und Veränderlichkeit von den großen bis zu den kleinsten Teilen eingeschränkt. Genau hier setzt das Prinzip

an: Einschränkung führt zu klareren Strukturen und Abläufen mit weniger möglichen Fehlerquellen. Abhängig von der Programmiersprache wird mehr oder minder möglich sein die Einschränkungen umzusetzen.

Das letzte Muster geht näher auf die Verknüpfungen zwischen den Komponenten ein. Den möglichen Informationsfluss festzulegen ist für einen einfachen Einsatz genauso wichtig, wie die einzelnen Komponenten selbst. Ein typischer Informationsfluss geht von der Benutzeroberfläche aus und wandert dann von der Interaktionsschicht in die tieferen Schichten bis meistens hin zu untersten Schicht, der Datenerhaltungsschicht, um dann wieder zurück zur Interaktionsschicht zu kommen. Ereignisse können Informationsflüsse auslösen und diese können in jeder Schicht auftreten. Wenn ein Ereignis auf der Interaktionsschicht ausgelöst wird, dann muss der damit verbundene Informationsfluss auch wieder dort landen. Nur dadurch kriegt der Nutzer eine Rückmeldung von dem System. Für die Umsetzung helfen dabei die bereits analysierten Informationsflüsse und die daraus entstandenen Vorgaben für die Schnittstellen der Komponenten untereinander.

IV.1.c. Implementierungskonzept des Softwareframeworks

Der Nutzer verwendet eigentlich eine kompilierte Version eines ITS eines Entwicklerteams von einem konkreten ITS-Projekt. Dies hat als Abhängigkeit das hier implementierte ITS-Softwareframework, welches wiederum auf der ITS-Softwarearchitektur aufbaut. Es gibt die vier Abstraktionsebenen in der Architektur. Die konkreteste ist die vierte Abstraktionsebene und damit auch am besten für eine konkrete Implementierung gedacht. Es müssen also alle vier Schichten mit den jeweiligen Subkomponenten und den jeweils dazugehörigen Verbindungen zu anliegenden Schichten in Programmcode transformiert werden. Das muss so gestaltet sein, dass sie sich leicht in einem neuen Projekt verwenden lassen können, die Grenzen der Softwarearchitektur einhalten und dabei ansonsten die Programmierer sehr frei in der Ausgestaltung der Komponenten sein können. Wenn jede Komponente implementiert wird und jede Komponente nur in exakt eine Abstraktionsebene angehört, dann kann der Aufbau des Softwareframeworks hierarchisch strukturiert werden, ohne vorerst die Schnittstellen zwischen den Komponenten zu berücksichtigen. Im zweiten Schritt werden dann die Verknüpfungen der Komponenten von einer Abstraktionsebene zur nächsten erstellt.

Die Software darf nicht nur aus einer Seite der Schichtenarchitektur betrachtet werden: Benutzeroberfläche oder Datenbanken. Eines der Fehler die häufig in bisherigen ITSs gemacht wurden: Es wurde sich nur auf die Datenbanken oder die Benutzeroberfläche fokussiert. Bei einer Schichtenarchitektur, bei dem die Benutzeroberfläche und die Datenbanken jeweils eine Schicht sind, könnte dies die Gleichberechtigung der Schichten untergraben. Es ist damit wichtig, dass eine Aktion nicht nur von der Benutzeroberfläche gestartet werden kann, sondern, dass auch die weiter unten liegenden Schichten selbstständig agieren und Handlungsketten auslösen können. Denn die Schichten sind nicht hierarchisch angeordnet sondern alle auf einer Ebene. Das heißt konkret, dass untere Schichten auch ohne einen dazugehörigen Aufruf aus einer höheren Schicht, Methoden der nächsthöheren Schicht zu den jeweils bekannten (und damit erlaubten) Komponenten senden darf. Es bedeutet nicht, dass die Kommunikation stattfinden muss. Sinnvolle Ereignisse zur Selbstaktivierung sind unter anderem Interaktionen des Benutzers, Änderung der Datenbank, eingehende Netzwerkdaten oder zeitbasierte Ereignisse. Eine weitere Einschränkung kann für den jeweiligen Fall sinnvoll sein und verletzt nicht die Architekturvorgaben. Jede Schicht hat seine Aufgabe und ist notwendig. Es ist wichtig dies bei der Implementierung des Softwareframeworks im Hinterkopf zu behalten, damit nicht eine Benutzeroberfläche um eine grandioses Datenbankverwaltungsprogramm geschrieben wird oder umgekehrt. Bei den Schichten ist es wichtig, das in beide Richtungen kommuniziert werden kann, dies aber nur mit der nächsthöheren und nächstniedrigeren Schicht mög-

lich ist. Es gibt mehrere Möglichkeiten einen Kommunikationskanal zwischen zwei Schichten zu ermöglichen. Die übereinanderliegenden Schichten haben jeweils eine Referenz zueinander. Dies kann durch ein Protokoll dazwischen modularisiert werden, sodass diese nicht miteinander gekoppelt werden, sondern mit einer anderen Implementierung der Schicht ausgetauscht werden kann.

In jeder Konkretisierung werden Entscheidungen getroffen, entweder indirekt durch die verwendeten Werkzeuge oder bewusst durch die gewählte Implementierung. Für so ein breit gefächertes Gebiet wie ITS wird es immer Spezialfälle geben, deren Implementierung von der einen oder anderen Veränderung verbessert werden könnten. Das Softwareframework bietet keine komplette Implementierung der Komponenten für einen Spezialfall an, weil dies zu stark von dem Einsatzzweck abhängt, ob es zum Beispiel ein Backend-Server oder eine lokale App ist. Da dies ein Softwareframework für praktisch umsetzbare ITS ist, wurde sich auch nach den bisher umgesetzten ITS gerichtet um Entscheidungen zu treffen, welche Bereiche, zu Gunsten einer allgemein besseren Verwendung, vernachlässigt werden können.

Umsetzungsanforderung: Komponenten als Protokoll und Klasse

Alle Komponenten werden vorerst als eine Protokoll im Programmcode definiert, die implementiert werden können. In Swift ist das Schlüsselwort dafür „protocol“ in Java heißt es „interface“. Zudem wird es auch für jedes Protokoll eine rudimentäre Implementierung geschaffen, damit man Komponenten vorläufig verwenden kann ohne sich gleich von Anfang an Gedanken über die Implementierung von allen Komponenten zu machen. Die Modularität und damit die Austauschbarkeit von Klassen ist dadurch wesentlich einfacher. Es gibt weniger Abhängigkeiten untereinander. Die Kopplung ist geringer. Eine Komponente kann ausgetauscht werden ohne dass eine andere geändert werden muss. Es wird weniger Wissen in den Programmcode integriert (hartkodiert), sondern wird in externen Ressourcen wie Datenbanken oder Textdateien gelesen. Als Beispiel wird es in der Persistenzschicht keine Zeichenkette mit einem Pfad zu einer Datenbank oder direkten Klassennamen der Implementierungen der Datenbankschicht geben können, weil man die Schnittstellen verwendet. Die Wiederverwendbarkeit einzelner Komponenten, die Lesbarkeit und der Wartungsaufwand wird dadurch verbessert. Swift erlaubt Protokolle mit Implementationen (wie in Java eine „abstract class“). Womöglich sollen diese genutzt werden, anstatt die rudimentäre Implementierung der Protokolle zu verwenden. Zum Beispiel beim verknüpfen der Komponenten untereinander. Die rudimentäre Implementierung der Protokolle kann dann diese Methoden einfach verwenden.

Welchen Typ die rudimentäre Implementierung hat ist eine weitere konkrete Entscheidung. Das Softwareframework stellt Typen und Funktionen zur Verfügung die in einer konkreten ITS-Implementierung verwendet werden können. Es gibt sechs Typen in Swift, zwei davon sind nur zusammengesetzte Typen ohne Namen die deshalb nicht verwendet werden können ("closure“ und „tuple“). Die restlichen Typen sind „enum“, „protocol“, „class“ und „struct“. Ein „enum“ ist semantisch eine Aufzählung und passt deshalb nicht. Ein „struct“ hat die semantische Zuweisung als Kopiervorgang, verwendet „call-by-value“ als Kopiervorgang bei Funktionsaufrufen, keine Vererbung und keine Identität der einzelnen Ausprägungen. Dies ist schon ein Problem bei einer komplizierten Mehrfachreferenzierung wie es in der Softwarearchitektur verwendet wird. Da ein „protocol“ keine gespeicherten Attribute besitzen kann, bleibt nur noch die klassische Klasse „class“ übrig.

Umsetzung der UML-Softwarearchitektur zu objektorientierter Programmierung (OOP)

Die Softwarearchitektur ist als ein UML-Komponentendiagramm geschrieben, welche an die jeweilige Programmiersprache angepasst werden muss. Dazu muss man die Semantik von UML verstehen. Es gibt zwei Beziehungen einer Komponente zu einer anderen. Die

Verschachtelung und die Schnittstelle. Die Verschachtelung ist eine Beziehung, bei der eine Komponente eine andere besitzt. Dies kann in der OOP und damit auch in Swift durch eine starke Referenz umgesetzt werden. Konkret in Swift bedeutet es, dass ein Objekt erst dann aus dem Speicher gelöscht werden kann, wenn es keine starken Referenzen mehr gibt. Es bedeutet im Umkehrschluss: Zu jedem Zeitpunkt existiert für diese Referenz eine Komponente. Für die Schnittstelle gibt es die bereitgestellte Schnittstelle und die erforderliche Schnittstelle. Diese hingegen erfordern keine Existenz des Gegenübers. Solch eine Beziehung kann mit einer schwachen Referenz umgesetzt werden, welche auch auf nichts zeigen kann. In Swift benötigt dies eine zusätzliche Änderung des Typs und spezielle Abfangmechanismen in der Verwendung solcher Referenzen.

Nehmen wir als Beispiel zwei Schichten mit jeweils drei Komponenten welche diese besitzen, siehe hierzu Abbildung R-IV1c-1. Jede Schnittstelle als Linie zwischen zwei Boxen wird zu zwei schwachen Referenzen. Jedes Besitzverhältnis wird zu einer schwachen und einer starken Referenz. Transformiert zu Objekten haben diese keine Verschachtelung.

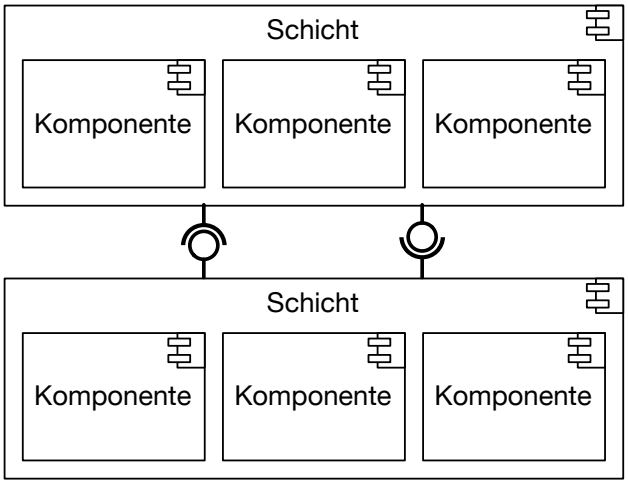


Abbildung R-IV1c-1: Beispiel einer Verschachtelung aus zwei Schichten

Nach der Transformation existieren nur noch Objekte und Referenzen, siehe dazu Abbildung R-IV1c-2. Welches das höhergestellte Objekt ist erkennt man meist daran, dass das höhergestellte Objekt die starke Referenz auf das andere Objekt hat. Letztendlich ist es aber die Relation zur erstaufgerufenen Quelldatei, die entscheidet welche Komponente höher einzuordnen ist. Bei Swift ist es die „main.swift“ die in einer Applikation immer vorhanden ist. In einem Framework existiert diese Datei nicht und jedes Projekt, dass das Framework verwendet, könnte theoretisch die Struktur zerpflücken und die Komponenten in einer neuen Implementierung anders als in der Architektur vorgesehen zu verwenden. Das kostet wesentlich mehr Arbeit als die vorhandene Aufteilung zu nehmen und mit wenig Code anzuwenden, soll aber als Möglichkeit offen bleiben für Weiterentwicklungen der Architektur ohne dass das Softwareframework gleich unbrauchbar wird. Wendet man diese Transformation für die gesamte Architektur an, so kann man alle Komponenten nur mit Referenzen implementieren.

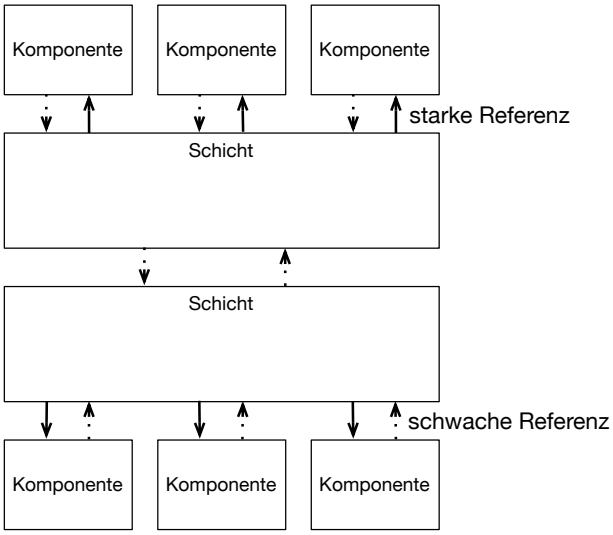


Abbildung R-IV1c-2: Referenzen zur Umsetzung von Verschachtelung in Swift

Zwei Varianten

Das Softwareframework wird in zwei Varianten vorhanden sein. Die erste Variante implementiert nur die Komponenten der niedrigsten Abstraktionsebene. Dies ist für kleinere Projekte gedacht oder für eine Architektur die weniger Komplex ist. Alle Komponenten sind auf der gleichen Ebene. Es gibt keine Verschachtelung und keine übergeordnete Funktionalität. Diese Architektur ist dann gut einzusetzen, wenn die Codebasis nicht so groß ist. Es gibt dann weniger Klassen und weniger Notwendigkeit für wiederverwendbaren Code oder die Verwaltung innerhalb von mehreren Komponenten. Die zweite Variante Implementiert alle Abstraktionsebenen, alle Komponenten und damit die komplette Softwarearchitektur. Für anspruchsvolles ITS ist die komplette Implementierung angeraten.

IV.1.d. Projekt im Detail

Für die Erstellung des Projektes wurde Xcode verwendet. Das kompilierte Softwareframework ist nicht nur auf macOS lauffähig. Das Projekt mit dem Quellcode hingegen ist ein XCODEPROJ-Datei. Es wurde sich für diese IDE entschieden weil dies der Standard des Betriebsystems macOS für Swift ist. Die Projektdatei beinhaltet dabei keinen Quellcode, sondern sie referenziert ihn nur. Für eine Strukturierung der Dateien wird ein Dateibaum mit Referenzen verwendet. Hierbei gibt es in Xcode zwei Arten um Dateien zu gruppieren: Mit Referenzen auf im Dateisystem existierende Ordner oder mit Gruppen die nur in Xcode definiert wurden. Damit die Ordnung, welche im Projekt vorkommen, auch im Dateisystem widergespiegelt werden, sind Referenzen auf Ordner die richtige Wahl. Alle Quelldateien und deren Hierarchie lassen sich damit auch ohne Xcode problemlos einsehen und editieren. Xcode ist für die vereinfachte Kompilierung der Quelldateien in eine Binärdatei. Dies lässt sich auch mit anderen oder komplett ohne IDE mittels der Kommandozeile und dem Kommandozeilenbefehl „swiftc“ in Bash auf macOS, Linux und Windows bewerkstelligen.

Für die Implementierung des Softwareframeworks in Swift wurde zunächst in Xcode ein neues Projekt angelegt. Es steht von Xcode die Wahl zwischen einer FRAMEWORK-Datei oder einer A-Datei zur Verfügung für ein Target als Softwareframework. Die A-Datei wird mittels des eines „LD"-Kommandozeilenbefehls verlinkt und des „AR"-Kommandozeilenbefehls zu einer A-Datei archiviert. Die A-Datei ist statisch und muss in jedem Projekt extra eingebunden werden. Sie beinhaltet nur die kompilierten Quellcode und Headerdateien (H-Dateien). Swift wird auf macOS, iOS, iPadOS, tvOS, watchOS und homeOS mit dem Betriebssystem mitgeliefert. Da Swift erst mit Version 5 stabil im „Application Binary Interface“ (ABI) ist und dies zu einem Betriebssystem passt welches 2019 benutzt wird,

muss die Swift-Runtime einer älteren Version bei einer A-Datei noch mit in der Binary mitgeliefert werden. Dies führt dazu, dass die A-Datei seine immer eigene Swift-Runtime mitliefern müsste. FRAMEWORK-Dateien hingegen sind dynamisch und benutzen die gleiche Swift-Runtime und können dem gesamten Betriebssystem zur Verfügung gestellt werden, sie ähneln damit den DLL-Dateien von Microsoft. Es ist zudem möglich in einer FRAMEWORK-Datei mehr als nur Quellcode zu integrieren, sondern Dateien jeglichen Formats. Deshalb wurde sich für eine FRAMEWORK-Datei als Zielformat für das Softwareframework entschieden. Selbst wenn man diese statisch verwendet, hat man immer die Option sie zukünftig dynamisch zu verwenden. Eine FRAMEWORK-Datei erlaubt außerdem Dateien hinzuzufügen die kein Quellcode sind. Von solch einer Fähigkeit wird zwar kein Gebrauch gemacht, trotzdem ist es gut auch hier die Freiheit zu haben.

Für das Projekt in Xcode wird eine „Info.plist-Datei“ benötigt. Dessen Aufgabe ist nur wenige Metadaten in einem simplen XML-Format in nur einer Ebene zu speichern. Darin enthalten sind der Dateityp, die Versionsnummern, Copyright und der Name der zu erstellenden FRAMEWORK-Datei. Damit lassen sich auch andere IDEs mit grundlegenden Daten über das Projekt füttern ohne dabei das internes Dateiformat von Xcode zu verwenden.

Wie in der Umsetzungsanforderung begründet werden alle Schnittstellen als ein Protokoll implementiert und die Anforderungen an die Typen, welche diese Protokolle implementieren sind einzig und allein, dass diese eine Klasse sein müssen. Damit sind die rudimentären Implementierungen automatisch auch Klassen die jeweils genau ein passendes Protokoll implementieren.

Um explizit die Schnittstellen in der Benennung von einer Implementierung zu trennen und es der Implementierung zu überlassen, ob sie die Komponente einfach mit dem Namen der Komponente der Architektur zu verwenden, bekommen alle Schnittstellen das Suffix „Protocol“. Der Nachteil ist eine Verlängerung des Namens um acht Zeichen.

Es ist wichtig, dass sich die Komponenten in der Erstellung und Initialisierung nicht zu stark unterscheiden. Als eine Einigung dafür, wenn man eine unbekannte Komponente bekommt, wie man sie einbindet und zum laufen bekommt. Damit deren Verwendung einheitlich ist, wird bei der Initialisierung keine bevorzugt. Wenn es zyklische Referenzen gibt, dann müssten alle Komponenten exakt gleichzeitig erzeugt werden um sicherzustellen, dass immer alle Komponenten zu jedem Zeitpunkt eine Referenz besitzen. Es gibt dabei drei Probleme. Erstens gibt es keine Option auch zur Laufzeit Komponenten austauschen zu können. Zweitens gibt es das Problem der gleichzeitigen Initialisierung. Drittens gibt es zyklische Referenzen, welche es nicht erlauben den Speicher von unbenötigten Objekten zu reinigen. Die Lösung ist: Diese Referenzen zu schwachen Referenzen zu ändern, dass heißt wenn eine Komponente entfernt wird, die Referenz automatisch auf eine Nullreferenz ("nil“ in Swift, „null“ in Java) geändert wird. Damit sind zyklische Referenzen kein Problem mehr, es lassen sich Komponenten nachträglich austauschen und die Initialisierung dieser Referenzen wird einfach ausgelagert. Das bedeutet die Referenzen müssen gesetzt werden können. Um auch nach der Initialisierung nachträglich von der nächsthöheren Instanz gesetzt zu werden. Wurden sie einmal gesetzt, so muss auch auf sie zugegriffen werden können, deshalb müssen sie auch lesbar sein. Da die Referenz auf beiden Seiten existieren muss, gibt es für jede Verknüpfung eine Referenz auf Seiten der einen Komponente zur anderen und umgekehrt. So verdoppeln sich zwar die Anzahl der Referenzen, aber es ist jeder Komponente möglich auf die Komponente der nächsthöheren Schicht zuzugreifen und damit auch auf gemeinsame Ressourcen und Funktionalitäten. Die Referenzen selbst haben den Typ des passenden Protokolls, sodass keine Kopplung zwischen den Implementierungen der beiden Komponenten entsteht. Jetzt müssen die Klassen welche das Protokoll adoptieren diese Referenzen in der Form implementieren und bei alle besitzenden Komponenten die Referenzen aktualisieren, sobald eine der abhängigen Referenzen aktualisiert wird. Das Protokoll und die dazugehörige rudimentäre

Implementierung befinden sich beide zusammen immer in einer Datei mit dem Namen der Komponente, so hat man gleich auf einen Blick beide Definitionen, wenn man den Quellcode von einem der beiden nachschlägt.

Da das Softwareframework selbst ein anderes Dateiziel hat für das es kompiliert wird, müssen Typen und Felder von Klassen, welche in einem anderen Kompilierungsprozess verwendet werden, erst explizit sichtbar gemacht werden. Dies geschieht mit dem Schlüsselwort „public“, an jede Protokolldefinition als auch an jede Klassendefinition und deren Felder innerhalb des Softwareframeworks.

Die Szenen haben nicht ohne Grund ein anderes Suffix als die restlichen Komponenten. Es ist immer nur eine Szene gleichzeitig aktiv und welche Szene als nächstes aktiv ist, entscheidet die Prozesssteuerungskomponente. Damit sich die Szenen von den anderen Szenen unterscheiden, um in einer Funktionalität bei der nur eine Szene erlaubt sein darf nicht irgendeine Komponente als Eingabe oder Ausgabe verwendet wird, sind diese mit einer anderen Endung gekennzeichnet. Swift interessiert der Dateiname nicht. Um auch Swift diesen Unterschied respektieren zu lassen, wurde ein gemeinsamer Typ geschaffen, ein Protokoll, dessen einziger Zweck es ist, die Szenen zu gruppieren. Dieses Protokoll hat den simplen Namen „SceneProtocol“. Dies ist insbesondere dann wichtig, wenn keine Schichten als eigenständige Komponenten vorhanden sind. Die Prozesssteuerungskomponente ist die Komponente, der diese Eingruppierung beim programmieren stark hilft, um viel einfacher und sicherer zu implementieren, denn sie ist die einzige Komponente, die in ihrer Funktionalität Referenzen von anderen Komponenten übergibt. Solch eine Gruppierung wird von keiner anderen Komponente als deren Grundaufgabe gesehen. Interessanterweise sind keine Fälle vorgekommen bei denen die Funktionalität nur für die anderen Schichten für spezielle Fälle benötigt wurden. Falls dies in einer Implementierung für andere Gruppierungen Sinn ergibt, dann ist es nur der Quellcode von einer Zeilen um solch ein gruppierendes, leeres Protokoll hinzuzufügen. Selbst die Zentralwissenkomponente übergibt keine Komponenten sondern greift nur darauf zu.

Variante 1: Nur die Komponenten der Abstraktionsebene 4

Die Variante 1 nimmt nur die Komponenten der vierten Abstraktionsebene. Die Ordnerstrukturierung des Softwareframeworks repräsentiert dennoch alle Ebenen, siehe dazu Abbildung R-IV1d-1. Protokolle und rudimentäre Implementierung sind jeweils in einer SWIFT-Datei. Meta-Projektdateien sind „ITS-Components.xcodeproj“ und die „Info.plist“. Zusammenfassendes Protokoll „SceneProtocol“ ist in „SceneProtocol.swift“. Wie schon angegeben soll die Auslagerung der Initialisierung der Referenzen zu den anderen Komponenten welche in der gleichen Abstraktionsebene sind in der nächsthöheren Instanz vorgenommen werden. Da es hier aber keine höhere Instanz gibt, wird in dieser Variante von eine Klasse implementiert, dessen Aufgabe nur genau das ist. Diese ist mit dem Namen „ComponentsConnector“ in „ComponentsConnector.swift“ definiert.

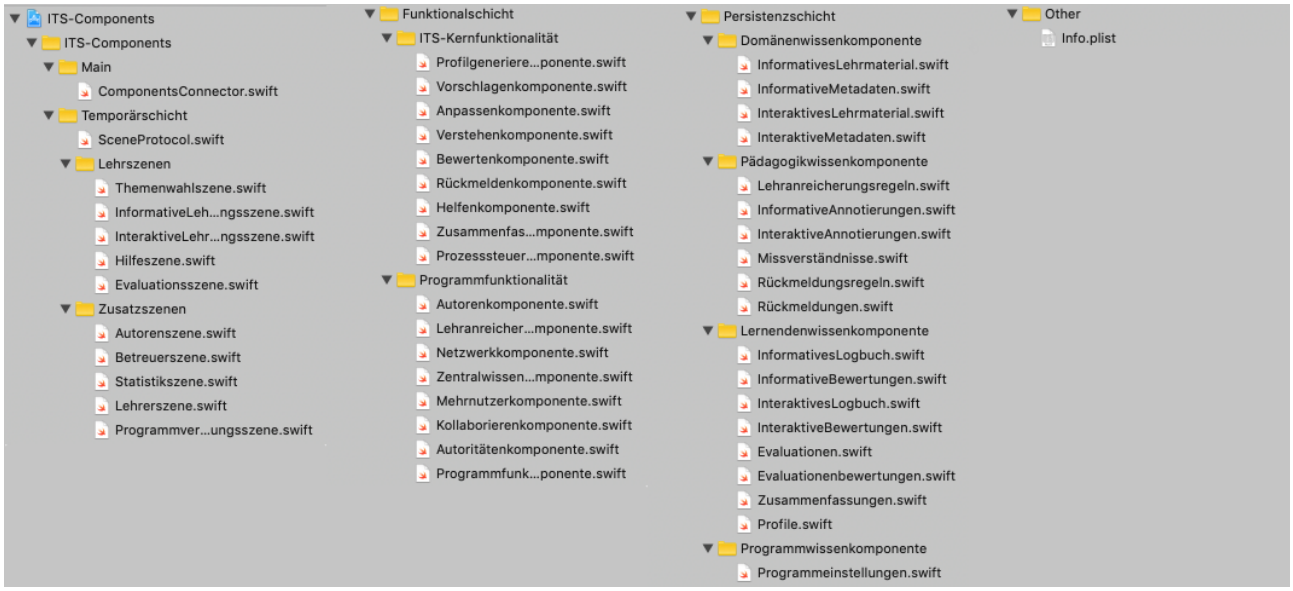


Abbildung R-IV1d-1: Ordnerstruktur des Projekts mit Variante 1

Variante 2: Komplette Softwarearchitektur

Die Variante 2 erstellt für jede Komponente in der Softwarearchitektur eine Datei mit Protokoll und rudimentärer Implementation als Klasse, also für Abstraktionsebene 1 bis 4. Die Ordner haben zwar die gleichen Ebenen, aber in dieser Variante befinden sich auch in den Zwischenebenen Dateien, die die Abstraktionsebenen über der vierten Abstraktionsebene repräsentieren, siehe Abbildung R-IV1d-2. Hier gibt es eine Klasse die das gesamte ITS abbildet. Die eigene Implementierung wird also innerhalb eines vorgegebenen Konstrukts integriert, anstatt wie bei Variante 1 den Einstieg und „Hülle selbst zu programmieren. Die Funktionalität des ITS wird durch eine Ausprägung der Klasse „ITS“ abgebildet, außerhalb sollte es keine weitern Objekte geben. Die einzelnen Aufgaben werden an referenzierte Objekte weitergereicht, die diese Ausprägung besitzt (starke Referenz).

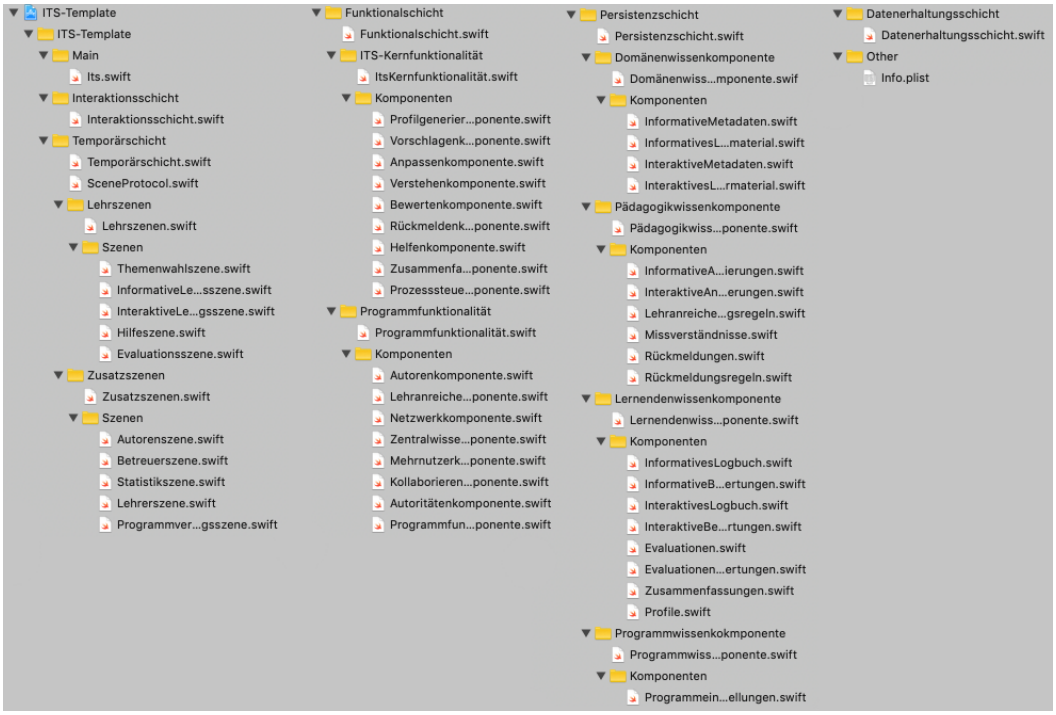


Abbildung R-IV1d-2: Ordnerstruktur des Projekts mit Variante 2

In dieser Variante gibt es verschiedene Abstraktionsebenen, also ist eine Components-Connector-Klasse nicht notwendig. Der Vorgang der Verknüpfung wird hier folgenderma-

ßen durchgeführt. Erst wird Bottom-Up ein Objektbaum erstellt mit den besitzenden Referenzen. Jede mehr abstrahierte Komponente kann sich also sicher sein, dass es die Komponenten gibt, weil sie direkt im Konstruktor übergeben wird und nicht mehr deallozieiert werden kann. Die benötigten Komponenten werden mit dem Konstruktor übergeben. Von Top-Down werden danach wieder die Komponenten von den fünf Schichten wieder verknüpft. Die Auslagerung der Initialisierung wird von der nächsthöheren Schicht bewerkstelligt, wenn dies geschehen ist wird die Verknüpfung der darunterliegenden Schicht in den darunterliegenden Schichten gestartet, bis bei den Komponenten der vierten Schicht alle Komponenten verknüpft sind. Die Aktualisierung geschieht durch eine durch eine Protokollerweiterung, welches eine Methode schon für alle Klassen des jeweiligen Protokolls vorimplementiert. Diese Methode wird „connectSubcomponents“ genannt und ist in allen Protokollen vorhanden, auch wenn sie in jedem Protokoll andere Referenzen miteinander verknüpft.

Codesignierung

Damit die Quelle des kompilierten Frameworks eindeutig ist wurde der Code beim kompilieren mit dem Namen der Autors dieser Doktorarbeit signiert. Jeder der also das Softwareframework verwendet kann es lesen, aber die Signatur kann nur erzeugt werden. Die Signatur ist mein Developer Account bei Apple, die auch das Rootzertifikat besitzen. So kann die Binary nicht ohne Verletzung des Zertifikats verändert werden.

IV.1.e. Verwenden des Softwareframeworks

Das Softwareframework selbst ist nicht ohne ein konkretes ITS-Projekt lauffähig. Als Beispiel wird hier der Ablauf erklärt wie das Framework als FRAMEWORK-Datei in ein neues Projekt in Xcode eingebunden wird. Es gibt zwei Orte an dem eine FRAMEWORK-Datei liegen kann um in einem Xcode-Projekt verwendet zu werden. Entweder innerhalb des Xcode-Projekts selbst oder außerhalb davon.

Das Hinzufügen eines Frameworks in ein Xcode-Projekt ist denkbar einfach. Die FRAMEWORK-Datei wird per Mauszeiger direkt in die Projekthierarchie in den Xcode „Project Navigator“ gezogen und entweder kopiert oder verschoben ①. Nachträglich wählt die Projektdatei (blaues Symbol) im „Project Navigator“ aus ② damit die Einstellungen des Projektes in dem XCODEPROJ-Dateiformat (welches ein XML-Schema ist) um in Xcode diese Einstellungen grafisch angezeigt zu bekommen. Werden die Einstellungen des Projekts angezeigt, dann das „Target“ zum jeweiligen Produkt anklicken ③. Nun können die Reiter für das Target durchgegangen werden, dort die Karteikarte „General“ anklicken ④. Hier können unter dem Menüpunkt „Embedded Binaries“ oder „Linked Frameworks and Libraries“ Softwareframeworks verlinkt oder eingebunden werden, abhängig davon ob das Framework nur verlinkt oder auch mit ausgeliefert werden soll. Frameworks in der Liste der „Embedded Binaries“ sollten auch in der anderen Kategorie erscheinen, sonst können sie zur Laufzeit nicht verwendet werden, selbst wenn sie sich als Datei in dem Order der kompilierten Swift-Binary befinden. Die schon in dem im Projekt referenzierte oder kopierte FRAMEWORK-Datei kann unter einen der beiden Punkte mit Klicken auf den „+"-Knopf in einer Liste angezeigt, ausgewählt und dann hinzugefügt werden ⑤. Die Verwendung einer FRAMEWORK-Datei in einer anderen IDE (Integrated Development Environment) ist ähnlich einfach.

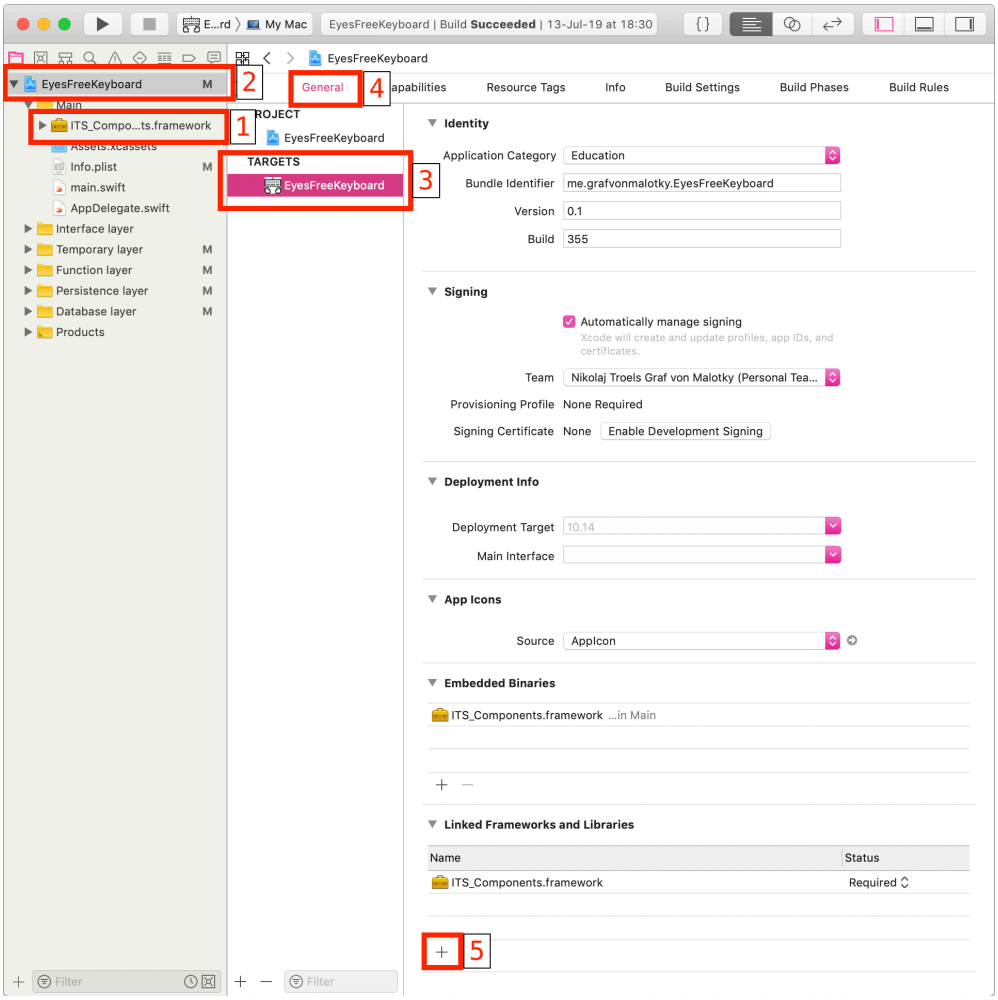


Abbildung R-IV1e-1: Screenshot mit Markierung der wichtigen Stellen in wichtiger Reihenfolge

Beim Einsatz des Frameworks gibt es zwei Varianten wo das Framework sich befindet. Einmal in dem ITS-Produkt statisch verlinkt und damit wurde es während beim Kompilieren in das ITS-Produkt kopiert. Die zweite Variante würde das Framework dynamisch linken und bei Ausführung des ITS-Produkts würde das Framework in bestimmten Pfaden des Betriebssystems gesucht werden. Da das Framework vom Speicherverbrauch sehr klein ist und es zusätzlicher Aufwand ist auf jedem System auf dem das ITS-Produkt laufen soll auch noch ein Framework zu installieren, ist die erste Variante stabiler, wartbarer und einfacher und sollte vorgezogen werden. In Xcode ist ein Framework statisch verlinkt wenn es in der Liste der „Embedded Binaries“ erscheint, siehe auch hier Abbildung R-IV1e-1. Falls von statisch auf dynamische Verlinkung geändert werden soll kann die Liste mit „-“ und „+“ editiert werden.

Nach Verlinkung des Softwareframeworks stehen alle Typen innerhalb des Projekts zur Verfügung. Dazu muss man nur das Framework selbst innerhalb der Dateien mittels einer Zeile am Kopf der Quelldatei importieren. Bei Swift wird der Import durch das Schlüsselwort „import“ gefolgt von dem Namen des Frameworks realisiert. Hier wären die folgend aufgelisteten Quellcodezeilen zu verwenden.

Für Variante 1: „import ITS_Components“.

Für Variante 2: „import ITS_Template“.

IV.2. Prototyp-ITS mittels des Softwareframeworks

Ein ITS wird mittels des mit dem vorgestellten Softwareframework erstellt. In Xcode wird ein neues Projekt erstellt, dass das ITS-Programm kompiliert. Das Softwareframework als FRAMEWORK-Datei wird als Abhängigkeit hinzugefügt wie bereits erklärt. Die äußeren Schichten werden durch konkrete Dateien abgebildet. Jegliche Funktionen die zusätzlich benötigt werden, können entweder von einer neuen Implementierung einer Komponente durch erben der rudimentären Implementierung oder neu implementiert werden und dabei die Richtlinien des jeweiligen Protokolls erfüllen.

IV.2.a. Ziel des Prototypen

Das Programm soll den Lernenden lehren, wie man ohne auf die Tastatur oder seine Finger zu schauen mit zehn Fingern nach einem vorgegebenen System schreibt. Das Ziel ist es, dass der Lernende nachträglich frei mit diesem System schneller und gesünder Texte schreiben kann. Tastaturschreiben ist dabei der Prozess, ein im Kopf vorgestelltes Wort durch Fingerbewegung angepasst an ein Tastaturlayout in Buchstaben auszudrücken, nicht umgekehrt. Bei dem Tastaturlayout gibt es zwei Einstellungen die relevant sind. Das Hardwarelayout und das Softwarelayout. Das Hardwarelayout, wie die Tasten physisch angeordnet sind und welche Größe und Form die Taste jeweils hat. Jede Taste sendet dabei nur einen Zahlencode, auch „key code“ genannt. Das physische Tippgefühl wird hier außer Acht gelassen, auch wenn es die Tippgeschwindigkeit beeinflussen kann. Das Softwarelayout wird vom Betriebssystem gestellt und bestimmt, wie die eingehenden „key codes“ interpretiert werden sollen, unabhängig davon, was auf der Tastatur auf den Tasten aufgedruckt ist. Es ist schwierig mehrere Layouts gleichzeitig zu trainieren, weil es schnell zu Verwechslungen kommt, deshalb wird sich nur auf ein Layout beschränkt. Es wird sich auf die ITS-Kernfunktionalität konzentriert werden, weil dies ein ITS ausmacht. In diesem Fall wird es also nur einen Lernenden pro ITS geben. Die Lehrstrategie basiert dabei auf eine langsame Steigerung des Inhalts des Lehrmaterials, insbesondere im interaktiven Lehrmaterial. Es wird erwartet, dass der Lernende bestimmte Dinge vergisst, dann wird entweder das Lehrmaterial reduziert oder bei sehr spezifischen Fehlern wird genau dies im Lehrmaterial vermehrt behandelt.

Konkret heißt es, dass der Lernende im informativen Lehrmaterial Dinge erklärt bekommt die langfristig zu einem effektiveren Tippen führen. Darunter fallen Themen wie die Motivation, Grundlagen der Haltung, Tipps für das schnellere Tippen, die Zugehörigkeit der Finger zu bestimmten Tasten, wie man das Karpaltunnelsyndrom vermeidet oder von vornherein häufig gemachte Fehler vermeidet, die nachträglich die Schreibgeschwindigkeit begrenzen. Das Wissen wird nicht genau wie erzählt abgefragt, sondern wie im Konzept definiert, durch einen interaktiven Teil angewendet in Szenarios, die näher an der Realität sind und trotzdem sich an den Lernenden anpassen. Für diese Anpassung werden die Interaktionen des Lernenden in einer Lehrsitzung gespeichert. Mit Hilfe des über den Lernenden gesammelte Daten kann einer Analyse herausfinden wo die Stärken und Schwächen bei dem Lernenden beim Tippen sind, in Hinblick auf unterschiedliche analysierte Faktoren. Das Ziel ist eine Fehlerquote von unter 5% bei gleichzeitiger Erhöhung der Tippgeschwindigkeit. Die Tippgeschwindigkeit bei Beibehaltung einer maximalen Fehlerquote kann auch durch diverse Maßnahmen erhöht werden, zum Beispiel zufällig Lehrmaterial auswählen und anpassen, auf die schnellsten getippten Tasten fokussieren, stupide die häufigsten Vorkommenden Reihenfolgen trainieren und so weiter. Die Bedingung, dass eine geringe Fehlerquote eingehalten werden muss verringert die möglichen Strategien des Prototypen. Dieser ITS-Prototyp implementiert, wie fast alle ITSs, nur eine Lehrstrategie. Falls ein anderes ITS mit ähnlicher Struktur aber anderer Lehrstrategie kreiert oder die Lehrstrategie nachträglich vom Prototypen angepasst wird, so ließe sich der

Lehrerfolg der verschiednen ITS durch ihre gleiche Basis gut vergleichen. Die Lehrstrategie dieses ITS-Prototypen konzentriert sich auf die Schwächen des Lernenden in Bezug auf einzelne Faktoren (wann und wie die Fehler gemacht werden) und versucht die Schwächen des Lernenden zu verbessern. Die personalisierten Faktoren des Lernenden, welche vorher aus dessen Daten herausgefunden wurden, werden für die Ausführung dieser Lehrstrategie benötigt, dass heißt, wenn die Schwächen bekannt sind muss der beste Weg für den Konter bekannt sein. Als Beispiel kann eine Methodik um einen Fehler zu kontern der in bestimmten Wörtern vorkommt sein, dass der Lernende nicht mehr einzelne Buchstaben im Wort korrigieren darf, sondern das gesamte Wort muss immer komplett neu geschrieben werden und dass in einer gewissen Zeitvorgabe. Das ist eine Anpassung die der Strategie entspricht, aber nur einen Teil dieser Strategie abdeckt, wenn bestimme Vorraussetzungen im Profil des Lernenden erfüllt sind.

Da die gesamte Wissensgrundlage eines blinden 10-Finger-Tippsystems im Vergleich zu anderen Domänen gering ist, hat das System zwei Lehraufgaben. Die erste Aufgabe basiert auf dem Verständnis des Tippsystems und der umliegenden Zusatzinformationen. Wie im Konzept festgelegt wird dies erst dann als verstanden vom System abgespeichert, wenn im interaktiven Lehrmaterial nachgewiesen wurde, dass der Lernende im Prototyp mittels der Messung der Tippgeschwindigkeit und Fehlerquote ein Mindesterfolg vorzuweisen hat. Der Schwierigkeitsgrad und die Komplexität nehmen dann bei steigendem Verständnis des Lernenden zu, indem die Anzahl der Tasten die getippt werden müssen didaktisch sinnvoll erweitert wird oder Modifikatoren hinzugefügt werden. Die zweite Aufgabe lehrt nicht unbedingt neues Wissen, aber verfeinert das Bekannte. Die Verbesserung der Tippgeschwindigkeit bei korrekter Ausführung ist dann keine neue Fähigkeit mehr die erlangt wird, sondern eine Feinjustierung der Fertigkeit, die gleichen Aufgaben viel schneller durchzuführen. Für eine Skalierung der Geschwindigkeit müssen vom Lernenden bestimmte Dinge beachtet werden, auf die spezifisch eingegangen werden kann.

An sich kann von diesem Programm nicht überprüft werden ob der Lernende nicht doch schummelt. Technisch wäre es zwar umsetzbar, zum Beispiel mit einer Kamera (oder sogar einer Infrarotkamera), welche die Handbewegungen beobachtet, aber praktisch für den Prototypen zu aufwändig. Die Handerkennung müsste zuverlässig bei vielen Tastaturen, Kameras und Hautfarben funktionieren, damit sich der Aufwand überhaupt lohnen würde. Selbst dann ist Schummeln nicht ausgeschlossen. In einem Labor mit festen Parametern für die Beleuchtung, Kamera und Tastatur mit speziellen Markierungen mag dies unter Umständen durchführbar sein.

Der Arbeitsaufwand des Prototypen ist zwar kein Kriterium das wichtig für dessen Entwicklung war, weil das vorrangige Ziel war eine Machbarkeit aufzuzeigen, dennoch ist es interessant diesen für sich zu betrachten. In Kapitel I.1.c wurde der Arbeitsaufwand von pro Lehrstunde auf 200-300 Stunden geschätzt, wenn eine domänenspezifische Autorenwerkzeuge zur Verfügung stehen sogar nur 50-100 Stunden. Der Prototyp wurde von einem Entwickler implementiert der gleichzeitig ITS-Experte, Softwareentwickler als auch der Domänenexperte ist. Durch das schon entwickelte Softwareframework wurde die benötigte Arbeitszeit gewiss maßgeblich reduziert. Da der Entwickler des Prototypen zusätzlich selbst der Entwickler des Softwareframeworks und der dahinter liegenden Konzepte ist lässt sich die Einarbeitungszeit in das Softwareframework und dessen Modelle schlecht einschätzen.

Der Prototyp selbst ist in einer Domäne bei dem die Aufgaben nicht statisch sein müssen sondern sich stark wiederholen und die automatische Generation von Lehrmaterial vergleichsweise einfach ausfällt. Insgesamt fand die Entwicklung des Systems und die Entwicklung des Lehrmaterials leider nicht getrennt statt. Das System wurde speziell auf diese Domäne hin optimiert, sodass sich die Entwicklung des Lehrmaterials und des Systems vermischten. Es wird grob geschätzt dass die Entwicklung über 100 Stunden gedauert hat. Das dedizierte Lehrmaterial um die theoretischen Grundlagen zu erlernen kann

in etwa einer halben Stunde durchgearbeitet werden. Geht man von dem ungefähren Ziel aus, dass 10-Finger-Systems mit einer Tippgeschwindigkeit von etwa 50 deutschen Wörtern pro Minute zu erlernen, kann bei täglichen kurzen Trainings von circa 30 Minuten dies in einem Monat erreicht werden, was ein automatisch generierten Lehrmaterial von 15 Stunden entspräche. Wenn man annimmt, dass die Hälfte der Entwicklungszeit in das Lehrmaterial investiert wurde, dann wäre das eine Geschwindigkeit von 1 Stunde Lehrmaterial auf 100 Stunden Arbeitszeit. Dabei wird vollkommen außer Acht gelassen, dass automatisch an den Lernenden angepasstes interaktives neues Lehrmaterial generiert werden kann. Da der Prototyp sein interaktiven Inhalt fortwährend anpasst und den Schwierigkeitsgrad erhöht gibt es nicht die Möglichkeit den Inhalt zu meistern. Der höchste Schwierigkeitsgrad ist unmöglich zu schaffen. Besonders durch die automatische Generierung von angepassten Lehrmaterial ist diese Entwicklung des ITS kosteneffizienter als statisches Lehrmaterial. Besonders mit dem Augenmerk auf die Anpassung auch nach Verschlechterung der Fertigkeit oder bei vergessen von Grundlagen beim Lernenden. Der Lernende muss nicht sich selbst einschätzen oder von vorne anfangen, sondern sein Profil wird automatisch bei Verwendung aktualisiert und dass dafür passende Lehrmaterial generiert, zum Beispiel bei dem Fall, dass der Lernende also später das Lehrmaterial wiederholen will, weil er etwas vergessen hat oder seine Fertigkeit nachlässt. Zudem erlaubt das ITS eine schnelle Erweiterung um weitere Lektionen. Es ist zwar kein Autorensystem vorhanden aber die Schnittstelle für Domänenexperten sind Textdateien die trivial zu bearbeiten sind. Zusätzlich kann die Auswertung des Lernenden vom System ihm oder den Lehrenden objektive Statistiken liefern. Für eine Erweiterung des Systems für die Zusammenfassung und Durchschnittsberechnung aller Lernenden, automatische Hochladen und Fernkontrolle und Hilfe ist im Softwareframework schon die Grundlage geschaffen worden.

IV.2.b. Aufbau des Prototypen

Der Prototyp verwendet das Softwareframework in der Variante 1, weil das Projekt mit nur einem Programmierer für ein ITS kleiner ausfällt, wird auch die Variante genommen die eher für kleinere Projekte gedacht sind. Mit dem Framework als Grundlage besteht schon eine Architektur für den Prototypen dessen grobe Aufteilung die Schichten sind. Diese Schichten werden folgend einzeln durchgegangen. Für jede dieser Schichten werden, für die Komponenten welche angepasst werden müssen, neue Klassen erstellt. Diese Klassen nehmen die Protokolle aus der Softwarearchitektur der jeweiligen Komponente an. Alle anderen Komponenten bleiben bei der rudimentären Implementierung und es wird die schon vorhandene Klasse aus dem Framework genommen. Die beiden äußeren Schichten benötigen immer eine neue Implementierung, da dort keine Komponenten vorhanden sind.

Die Konkretisierung der Architektur zu einer Applikation lässt es nun zu, Komponenten für die beiden äußeren Schichten zu erstellen. In der Interaktionsschicht sind dies die XML-Beschreibungen der konkreten Oberflächendesigns der Ansichten. Genau das sind die Dateien im STORYBOARD-Dateiformat. Die Ansichten sind wie Vorlagen aufgebaut. Sie bestehen aus einem Design, bestehend aus der Definition der Elemente in der Ansicht, ihrer Position und ihr Aussehen und können auch Animationen enthalten. Diese Vorlagen haben Stellen die veränderlich sind, welche noch mit Werten gefüllt werden müssen. Eine STORYBOARD-Datei kann dabei viele Ansichten beinhalten. Der Einfachheit halber und damit jede Datei einen klar abgetrennten Verantwortungsbereich hat, herrscht in diesem Prototyp ein 1:1 Verhältnis. Die Oberfläche ist mit AutoLayout entwickelt, es ist eine deklarative Definition der Ansicht die sich dynamisch anpasst. Das bedeutet, die exakten Größen der Oberflächenelemente sind nicht statisch definiert, stattdessen gibt es Bedingungen die eingehalten werden müssen. So kann sich die Oberfläche jederzeit

schnell neuen Fenster oder Bildschirmgrößen anpassen oder auf mobilen Geräten eingesetzt werden. Am schwierigsten war das Design der Ansicht für die Interaktive Lehrszene. Da dort die Konzentration des Lernenden stark beansprucht wird durch den zeitlichen Druck schnell zu tippen. Dies wird durch eine visuelle Repräsentation geschaffen, die schneller aufgenommen wird, als Texte, die gelesen und dann verstanden werden müssen.

Um die Werte für jede einzelne Ansicht zu füllen gibt es Klassen die genau das machen, sie werden ViewController genannt und sind speziell auf die Bedürfnisse und das Wissen über eine Ansicht ausgelegt. Sie dienen als Verbindung vom Code zu den STORYBOARD-Dateien. Die ViewController werden jeweils von einer Szene gruppiert, welches die Funktionalität beinhaltet, dass für alle ViewController notwendig ist. Da die Szene häufig ähnliche ViewController und damit Ansichten verwaltet. Sowohl die ViewController als natürlich auch die Szenen befinden sich in der Temporärschicht. Um die Anforderungen der Applikation zu erfüllen, wird die Programmverwaltungsszene, die Statistikszene, die Autorenszene und die Szenen aus den Lehrszenen: Themenwahlszene, Informative Lehrsitzungsszene, Interaktive Lehrsitzungsszene, Evaluationsszene implementiert.

Die Funktionalschicht übernimmt die meiste Arbeit für die business logic, der Teil der Daten transformiert und auswertet. Die Prozesssteuerungskomponente ist dabei sehr wichtig um von einer Szene zur nächsten zu kommen. Für den Prototypen werden auch viele andere der Komponenten der ITS-Kernfunktionalität benötigt, weil dazu die Benutzeroberflächen mit den Szenen, die schon erzeugt wurden vorhanden sind. Es gibt eine Generierung von passiven Lehrmaterial, bei dem Textbausteine kombiniert werden, um einen langen Text für den informativen Teil der Lehrsitzung zu erhalten. Das interaktive Lehrmaterial wird auch generiert. Die Generierung fällt einfach aus, weil schon eine statische Liste von Zeichen die für eine Tastenkombination gibt. Entweder kann daraus eine unendliche Folge generiert werden oder es wird die Liste an Wörtern genommen. Die Wahl der richtigen Zeichen und Wörter hingegen fällt schwieriger aus. Schließlich soll der Lernende nicht irgendwas lernen, sondern seine Schwächen sollen verbessert werden. Deshalb wird das Profil nach aktiven Regeln durchsucht, die für feste Aktionen stehen die das ITS zur Auswahl verwendet. Einflussfaktoren für die Anpassung ist das Profil des Lernenden über Schwierigkeitsgrad und Zeichen beziehungsweise Zeichenfolgen die dieser schlecht beherrscht und die zu dem Schwierigkeitsgrad passen. Dazu muss die Funktionalschicht semantisch verknüpft haben was ein Schwierigkeitsgrad ist. Die Funktionalschicht benötigt das Verständnis was eine Taste ist, was eine Tastatur ist, was eine Hand ist, welche key codes und Tasten in einer Tastatur sind, welche davon Zeichen ausgeben, wie die Tasten durch Modifikatoren in ihrer Funktion verändert werden und so weiter. Damit kann die Anpassenkomponente und die Bewertenkomponente die richtigen Entscheidungen treffen. Die Bewertenkomponente kann damit die Eingaben richtig zuordnen und dann auf Korrektheit überprüfen, weil semantisch klar ist was das Ergebnis sein sollte und ob zum Beispiel der key code für die Taste „rechte Umschalttaste“ und der key code der zeitlich verzögert zu der Taste „1“ passt korrekt ist für das Zeichen „!“. Das wird insbesondere dann wichtig, wenn Tasten genommen werden, die keine Zeichen ausgeben, weil dann kein Textfeld nach der Eingabe abgefragt werden kann.

Die Persistierungsschicht bildet die Schnittstelle des ITS-Modells der Daten hin zu einer effizienteren Umsetzung, die abhängig von der Platform und den Zielen anders ausfallen kann. Die Klassen in der Persistierungsschicht halten sich an die Aufteilung der Komponenten.

Das Domänenwissen beinhaltet dabei kleinere Abschnitte über die Erklärung der Tastatur, wie die Standardstellung der Finger auf der Homerow ist, welcher Finger für welche Tasten da ist, wie das mit den Modifikatoren funktioniert, Motivation für die Geschwindigkeits- und Ergonomievorteile, ein wenig Geschichte über die Tastaturlayouts und Tippsysteme, Hard- und Softwarelayout und generell die Regeln des vorgestellten Tippsystems.

Diese einzelnen kurzen Texte lassen sich abhängig vom Wissensstand des Lernenden kombinieren um damit den Text für die Lehrsession in Inhalt und Länge zu variieren. Zudem sind hier die Regeln für das Verständnis, welche Tastenkombination welches Symbol erzeugen und welche Funktionstasten zwar ein Symbol haben, aber keines Ausgeben, zu welcher Hand die Tasten gehören, welche Keycodes die Tasten haben und im Layout dann verbunden werden.

Das Pädagogikwissen hat die Annotationen des Wissens, um nachträglich entscheiden zu können, wann welches Lehrmaterial am besten geeignet ist. Darin enthalten ist ein Schwierigkeitsgrad für die einzelnen Tasten und Textstücke. So bildet der Schwierigkeitsgrad eine Begrenzung der Wahl des Inhalts. Zum Beispiel die Abhängigkeit einer Taste zu anderen: Umschalt erst nach mindestens dem Verständnis der der Alphanumerischen Tasten, Caps lock erst nach Verständnis von Umschalt nach Shift, Zeichen der Alphas vor den Zahlen, Wörter nach Leertaste und Buchstaben, Backspace nach Homerow und so weiter. Auch ist in diesem Wissen welche häufig gemachten Fehler am besten zu vermeiden sind und wie das System sich dann verhalten sollte. Hier sind die vorgefertigten Antworten mit Ermutigung und Tipps für bestimmte Fehlerquoten enthalten, als auch Lob bei überdurchschnittlicher Leistung, die jeweils für die Taste und die Leistung auch inhaltlich angepasst werden.

Das Lernendenwissen ist am Anfang leer und wird erst mit den Interaktionen des Lernenden mit dem System während der Lehrsitzung und den Analysen davon gefüllt. Darunter fällt, welches informative Lehrmaterial wie häufig und wie lange gesehen wurde, welches interaktive Lehrmaterial gemacht wurde und wie die Fehlerquote war und wieviel Unterstützung dabei angezeigt worden ist. Konkret ist es jede Tastenkombination mit Zeitstempel die getippt wurde seit Beginn mit der zusätzlichen Information welche Tastenkombination für ein gewähltes Zeichen (auch mit Zeitstempel) gefragt war, inklusive der Modifikatoren und Funktionstasten. Dazu passend eine einzigartigen Nummer für Lehrsitzung und das Wort in dem der Buchstabe vorkam. Die Basisdaten die erhoben werden bei dem interaktiven Teil der Lehrsitzung sind überraschend klein, wenn man bedenkt welche Werte sich davon analysieren lassen. Die Zusammenfassung kann leicht mehr als das doppelte an Attributen daraus berechnen die in Statistiken oder dem Profil verwendet werden. Zum Beispiel reicht der Zeitstempel und in welchem Wort es vorkam aus, zuzuordnen welche Wörter an welchen Stellen mit identifizierbaren quantifizierbaren Fehlern der Lernende zu kämpfen hatte.

Das Programmwissen ist absichtlich minimal und speichert höchstens globale Einstellungen des Prototypen.

Die Datenerhaltungsschicht interessiert sich nicht für das Modell des ITS. Sie versucht nur den besten Weg der Speicherung zu finden. Es wurden zwei Varianten für die Speicherung der Daten gewählt. NoSQL-Textdokumente und SQLite als relationale Datenbank. Es sollten beide Varianten in einem Prototypen probiert werden. TXT-Dateien lassen sich von Autoren einfach editieren und SQLite ist sehr performant bei vielen Einträgen. Die Datenerhaltungsschicht teilt sich damit in die Datenbanken selbst und in den Code für die Verwaltung der Daten. Der Code ist in zwei Klassen, einen Datenbankmanager für den Umgang mit den Textinhalten und einen für den Umgang mit der SQLITE-Datei. Es war sehr wichtig, dass diese effizient gespeichert wurden, sodass eine Tastenkombination und der Zeitstempel nicht als Zeichenkette gespeichert werden konnte, sondern stattdessen eine Reihe von Zahlen und Wahrheitswerten genommen wurden. Auch wenn die Editierung durch die unterschiedlichen Modelle für Analysezwecke von Lehrforschern damit erschwert wird. Die Tabellen und Spalten für die Daten wurden aussagekräftig benannt, um dem entgegen zu wirken. Die SQLite-Datenbank speichert alle Interaktionen des Lernenden innerhalb der Lehrsitzungen. Das gestaltete informativem Lehrmaterial ist in kleine Abschnitte unterteilt, bei dem die Reihenfolge und Absätze relevant sind und die Anzahl der Absätze angepasst werden kann.

Ist in der Projektdatei eine native Applikation als Kompilierziel eingestellt, dann reichen die Datenbankdateien, die Dateien der Ansichten, der Code für das ITS und seinen Komponenten, die Datei für das Framework und die Projektdatei selbst nicht aus für eine Applikation. Benötigt werden für eine native App die „main.swift"-Datei, eine Klasse die vom Typ „NSApplicationDelegate“ ist, Ressourcendateien, eine Fensterdefinition und die „Info.plist"-Datei. Die Hierarchie der Ordner im Projekt entsprechen den Schichten, sonstigen Ressourcen und der Hauptgruppe „Main“ mit den für die App wichtigsten Dateien: „main.swift“, „Info.plist“ und so weiter. Das Softwareframework wurde in die Hauptgruppe kopiert, welche dann mit in die App kopiert wird. Also eine Statische Verlinkung des Softwareframeworks, damit es immer mit der App zusammen gebündelt ist und ausgeliefert wird. Es besitzt also keine Abhängigkeiten an Softwareframeworks die nicht im Betriebssystem oder in der Applikation selbst schon vorhanden sind. Die „Info.plist"-Datei hat wie beim Framework die Aufgabe wichtige Metadateien unabhängig von der Projektdatei zu speichern, die auch nach dem kompilieren erhalten bleiben und dem Betriebssystem und externen Programmen erlauben, diese Daten schnell und einfach zu lesen ohne die Binärdatei der Applikation zu öffnen. Das PLIST-Dateiformat steht für „Property list“ und ist eine spezielle XML-Version mit weniger Funktionalität, welches sich auf das Speichern von Werten ("properties“) beschränkt. Einstiegspunkt einer nativen Applikation ist immer die „main.swift"-Datei. In ihr gibt es keine Definitionen, sondern nur imperativ global ausgeführte Ausdrücke, ob Zuweisung, Deklaration oder Aufruf. Wenn der Prototyp gestartet wird, dann wird die „main.swift"-Datei ausgeführt. Diese erzeugt ein AppDelegate-Objekt und kann dann die Applikation erzeugen, die referenzzählende Speicherverwaltung ("AutoreleasePool“) und beides gemeinsam starten. Es wird nicht der Standard „NSApplicationDelegate“ verwendet, weil dieser zum Start gar nichts macht, außer das Standardfenster zu laden. Es wurde ein angepasster „AppDelegate“ in „AppDelegate.swift“ definiert der die ITS-Ausprägung erstellt und verknüpft. Wichtige Events zur Laufzeit der Applikation werden an Methoden des AppDelegate delegiert. Das angepasste AppDelegate-Objekt bekommt dann, wenn das Betriebssystem bereit ist die Meldung, dass die App in den Speicher geladen wurde. Hier kommt das Softwareframework ins Spiel. Es werden von jeder Komponente des ITS eine Ausprägung erzeugt mit entweder der angepassten oder der rudimentären Version. Alle Komponenten werden mittels des „ComponentsConnectors“ aus Version 1 des Softwareframeworks verknüpft. Ein Fenster und die Menüleiste wird in einer dafür angelegten „Main.storyboard“ definiert. Jetzt wird genau das Fenster vom AppDeletegate erzeugt und dann die Prozesssteuerungskomponente gefragt welche Szene als erstes die Kontrolle übernimmt und das Fenster füllt. Die erste Szene kann sich dann darum kümmern die passende Ansicht anzuzeigen. Da diese Szenen unter Umständen Bilder benötigten kommt hier die nächste Datei ins Spiel. Alle Bilder für die Benutzeroberfläche werden in der „Assets.xcassets"-Datei als Bundle zusammengefasst. Damit können sie direkt im Code als Objekte als auch in den Ansichten als vorhandene Elemente verwendet werden.

Um den konkreten Anforderungen gerecht zu werden wurden einige Komponenten neu implementiert ohne ihre Definition oder Verknüpfungen zu ändern, zusätzlich wurden einige Komponenten neu hinzugefügt. Die Komponenten der konkreten äußeren Schichten (Interaktionsschicht und Datenerhaltungsschicht) müssen dabei zu Komponenten der anliegenden Schicht Verknüpfungen besitzen während neue Komponenten der anderen Schichten nur innerhalb ihrer Komponente Verknüpfungen haben können. So wird vermieden die Architektur und ihre Beschränkungen einfach mit neuen Komponenten zu umgehen. Wie schon in diesem Kapitel aufgezählt und in Abbildung R-IV1e-1 zu sehen gibt es wenige Dateien die sich außerhalb der Architektur befinden und nur Metadaten zur Lauffähigkeit des Programms bereitstellen. Alle anderen Dateien gehören zu den jeweiligen Komponenten der Architektur, siehe hierzu Abbildung R-IV2b-1 welche deutlich zeigt

welche Komponenten von dem Softwareframework übernommen wurden und wie viele davon eine neue Implementierung brauchten.

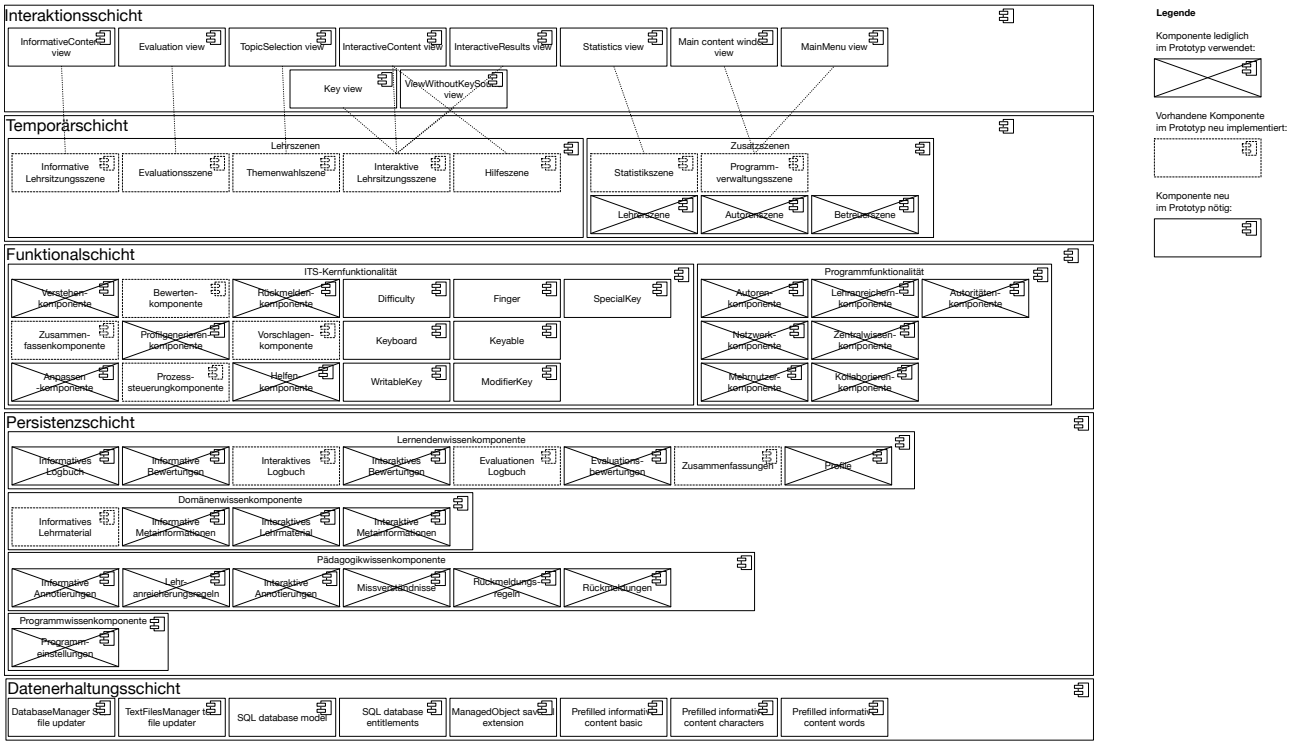


Abbildung R-IV2b-1: Komponenten des Prototypen

IV.2.c. Analyse des Lernenden im Prototypen

Die Eingaben des Lernenden werden wie vom Konzept definiert in zwei Schritten analysiert. Eine Zusammenfassung seiner Aktivitäten in den jeweiligen Lehrsitzungen enthält sowohl die direkt zur Laufzeit aufgenommenen Daten als auch noch daraus nicht interpretierte ableitbare zusätzlichen Daten die folgend aufgelistet sind.

Daten über die Gesamtleistung:

- Anzahl aller Lehrsitzungen
- Anzahl aller Lehrsitzungen (kategorisiert nach Zeit)
- Anzahl aller Buchstaben-Lehrsitzungen
- Anzahl aller Wörter-Lehrsitzungen
- Anzahl der Buchstaben-Lehrsitzungen die insgesamt abgebrochen wurden
- Anzahl der Wörter-Lehrsitzungen die insgesamt abgebrochen wurden
- Abbruchquote der Lehrsitzungen
- Durchschnittliche Fehlerquote
- Durchschnittliche Länge der Lehrsitzungen
- Durchschnittliche Tastenkombinationen pro Minute

Daten über die jeweiligen Lehrsitzungen:

- Anzahl an Tastenkombinationen die in der Lehrsitzung getippt wurden
- Durchschnittliche Tastenkombinationen pro Minute in der Lehrsitzung
- Fehlerquote an Tastenkombinationen die insgesamt in der Lehrsitzung getippt wurden
- Fehlerquote in Bezug zur verbleibenden Zeit in der Lehrsitzung (kategorisiert nach Sekunden)
- Fehlerquote in Bezug zum Wochentag (kategorisiert nach Wochentag)
- Fehlerquote in Bezug zur Uhrzeit (kategorisiert nach Stunden)

- Fehlerquote in Bezug zu der Anzahl an Lehrsitzungen in den vergangenen Stunden (kategorisiert nach Stunden bis 24)
- Fehlerquote in Bezug zu der Dauer der aktuellen Lehrsitzung (kategorisiert nach Minuten)
- Fehlerquote in Bezug zum zeitlichen Abstand zur letzten Lehrsitzung (kategorisiert nach Tagen)

Daten über die Tastenkombinationen:

- Dauer bis die richtige Tastenkombination gedrückt wurde (kategorisiert nach Tastenkombination)
- Fehleranzahl bis die richtige Taste gedrückt wurde (kategorisiert nach Tastenkombination)
- Anzahl an Tastenkombinationen die insgesamt getippt wurden
- Fehlerquote an Tastenkombinationen die insgesamt getippt wurden
- Anzahl die jede Tastenkombination getippt wurde (kategorisiert nach Tastenkombination)
- Fehlerquote die jede Tastenkombination getippt hat (kategorisiert nach Tastenkombination)
- Anzahl die jeder Finger an Tastenkombination getippt hat
- Fehlerquote die jeder Finger an Tastenkombination getippt hat
- Anzahl die jede Hand an Tastenkombination getippt hat
- Fehlerquote die jede Hand an Tastenkombination getippt hat
- Anzahl die jede Wortes an Tastenkombination getippt hat
- Fehlerquote die jede Wortes an Tastenkombination getippt hat
- Welche Tastenkombination vor einem Fehler in jedem Wort getippt wurde (kategorisiert nach Tastenkombination)

Es können noch viel mehr Daten erhoben werden in Bezug zu Wetter, subjektiver Einschätzung von Fitness, Gesundheit und Laune, was vorher gemacht wurde, ob andere das gleiche gemacht haben und was deren Erfahrungen im Vergleich waren und so weiter. Für die subjektive Einschätzung wird durch die Fragen nach einer Lehrsitzung eine Evaluation erhoben. Diese wurde in dem Prototypen einfach gehalten mit vier Fragen:

- „Wie war der Schwierigkeitsgrad der Lehrsitzung?“
- „Wie war die Länge des Textes Präsentation des informativen Lehrmaterials?“
- „Wie war die Länge der Interaktion des interaktiven Lehrmaterials?“
- „Wie bewerten Sie die Lehrsitzung insgesamt?“

Da der Prototyp nicht in der Lage ist großen Änderungen an der Darstellung zu bewerkstelligen in der Anpassung des Lehrmaterials, werden die Fragen nur so gestellt, dass sie auch zu einer Umsetzung führen können. Für Forschungszwecke kann es sehr wohl sinnvoll sein mehr Daten zu erheben als es dem Programm möglich ist sinnvoll auszuwerten. Diese wurden für diesen Prototypen aber außer Acht gelassen. Die Daten die automatisch gesammelt werden und nicht aus den Interaktionen des Nutzers mit dem Prototypen kommen, müssten erweiterten Zugriff der Applikation im Betriebsystem haben. Ohne direkten Verarbeitungszweck ist das eine heikle Datenschutzfrage.

Aus den gesammelten Daten wird ein Profil erstellt. Dies geschieht durch das Überprüfen der Zusammenfassung mittels festgelegter Regeln. Das Profil hat eine Ansammlung von festen Aussagen die dann gültig werden, wenn eine Regel zutrifft. Darunter fallen Regeln die folgend aufgezählt werden.

- Gesamte Fertigkeit abhängig von der Fehlerquote über alle Tasten, Bewertung wird erst vorgenommen wenn eine Taste 10 mal angeschlagen wurde.
- Problem eine bestimmte Taste zu tippen, Bewertung wird erst vorgenommen, wenn die Fehlerquote unter ein Limit fällt
- Problem eine Tastenfolge zu tippen, Bewertung wird erst vorgenommen, wenn die Fehlerquote unter ein Limit fällt

Einige Aussagen können immer getroffen werden, der Inhalt ändert sich dann nur abhängig davon, wie hoch die dazugehörigen Werte ausfallen. Andere Aussagen werden nur dann getroffen, wenn eine bestimmte Bedingung eintritt, da sie sonst uninteressant für die Anpassung werden. Feste Aussagen wären dann zum Beispiel: „Du beherrschst die Tasten der Homerow sehr gut. Du hast viele Fehler der Tasten unterhalb der Homerow. Du hast Probleme mit der Zeichenfolge 'ch'. Jede dieser Profilregeln hat eine eigene ID. Diese ID wird dann als aktiv gespeichert und lässt sich nachschlagen. Somit kann das Profil von einem Menschen verstanden werden, weil dieser die Texte liest, als auch effizient von einem System semantisch eingeordnet werden, weil die ID und deren Folgeaktionen bekannt sind. Der Mensch kann die Texte selbstständig auswerten und sich überlegen was er machen kann um sich dahingehend zu verbessern. Das System wird einfach so programmiert, dass eine bestimmte ID zu einer Veränderung des Verhaltens in diskreten Schritten führt.

IV.2.d. Adaption an den Lernenden im Prototypen

Die Anpassung des Prototypen an den Lernenden findet in drei Bereichen statt: Im informative Lehrmaterial, im interaktiven Lehrmaterial und in der Statistikansicht.

Das informative Lehrmaterial besteht im Domänenwissen aus kleinen Wissenselementen aus Text. Der angezeigte Text in Kapiteln richtet sich nach dem Profil des Lernenden. Der Schwierigkeitsgrad bestimmt die wählbaren Textbausteine. Wenn der Schwierigkeitsgrad steigt, können mehr Textbausteine für das informative Lehrmaterial verwendet werden, beim maximalen Schwierigkeitsgrad sind es dann alle Textbausteine die zur Verfügung stehen. Der Lernende hat aber auch einen Fortschritt welches die Textbausteine verkürzt oder gar nicht erst anzeigen lässt. Es gibt zwei Faktoren die die Textbausteine reduziert. Ob das Thema schon im informativen Lehrmaterial dem Lernenden präsentiert wurde und ob er eine hohe Fertigkeit in dem Thema im interaktiven Lehrmaterial mit einer geringen Fehlerquote und einer hohen Tippgeschwindigkeit unter Beweis gestellt hat. Die geforderten Tastenkombinationen sind selbst in Schwierigkeitsgrade eingeteilt. So bekommt ein Lernender der bestimmte Textbausteine schon gelesen hat und einer hohen bewerteten Fertigkeit in Themen der ersten Schwierigkeitsgrade wenig bis gar keine Textbausteine zu sehen, obwohl diese für das gewählte Thema mit inhärenten Schwierigkeitsgrad passend ist.

Die Anpassung des interaktiven Lehrmaterials, abhängig vom Lernenden, findet kontinuierlich statt. Die gewählten Buchstaben und Wörter sind abhängig von der Buchstabenreihe die bisher beherrscht werden und welche Buchstaben besonders schlecht sind. Anstatt nur immer wieder den gleichen Buchstaben oder das gleiche Wort welches am besten zum Lernenden passt zu nehmen, muss eine Abwechslung herrschen. Anpassung der Farbe der Geschwindigkeit und Genauigkeit abhängig vom Vergleich zum Durchschnittswert. Grün heißt in den westlichen Ländern gut und rot heißt schlecht. Keine Einfärbung oder weiß heißt neutral. Da die Eingabe eine hohe Konzentration des Lernenden erfordert, kann eine textuelle Rückmeldung mit langen Texten während der interaktiven Lehrsitzung nicht richtig verarbeitet werden. Eine zusätzliche Anpassung kann nicht nur eine Rückmeldung per Textantwort sein, sondern auch eine Hervorhebung von vorhandenem Inhalt durch wie hier geschehen einer Einfärbung. Die Adaption an den Lernenden erfüllt auch deshalb die Anforderungen eines ITS, weil sie kontinuierlich ist, nicht nur zur Erstellung der Lehrsitzung verwendet wird, sondern abhängig vom Verhalten des Lernenden zur Laufzeit ist.

Die Statistikszene ist der Bereich des Prototypen bei dem die Analysen des Lernenden zusammenkommen. Es werden Diagramme angezeigt die die wichtigsten Attribute der Zusammenfassungen grafisch aufzeigen. So kann der Lernende aus der Visuellen Darstellung Schlüsse über seine Entwicklung in den Bereichen einsehen. Es ist damit möglich

Rückschlüsse auf besondere Schwächen und Stärken zu ziehen und sich selbst Pläne für den weiteren Fortschritt zu machen. Das System selbst wertet auch die Zusammenfassungen aus und die Analysen aus dem Profil werden unter den Tipps angezeigt. Dort gibt es in Textform eine Bewertung vom Lernenden und auf herausstechende Eigenschaften wird hingewiesen. Mit Tipps wie man den Schwächen entgegenwirken kann vollendet das System seine Analyse ab.

IV.2.e. Benutzung des Prototypen

Das entwickelte Programm ist eine Applikation die nativ auf macOS ausgeführt werden kann. Zum starten der Applikation braucht man die ausgelieferte APP-Datei einfach nur doppelt anzuklicken und schon startet das Hauptmenü, siehe Abbildung R-IV2e-1. Dort gibt es vier Menüpunkte die ausgewählt werden können: Starten einer Lernsitzung, Statistik & Tipps, Autorenwerkzeug und die Einstellungen. Man kann die Applikation auch komplett mit der Tastatur oder der Stimme steuern, dass verbessert die Barrierefreiheit. Mit Tab lässt sich zwischen den Elementen wechseln und mit der Eingabetaste bestätigen, die Auswahl ist dafür durch einen rosa Markierungseffekt hervorgehoben. Jedes Element hat intern einen eigenen Namen und eine Beschreibung die für die Steuerung mit der Stimme verwendet wird. Zusätzlich gibt es die Applikation in hell und dunkel.

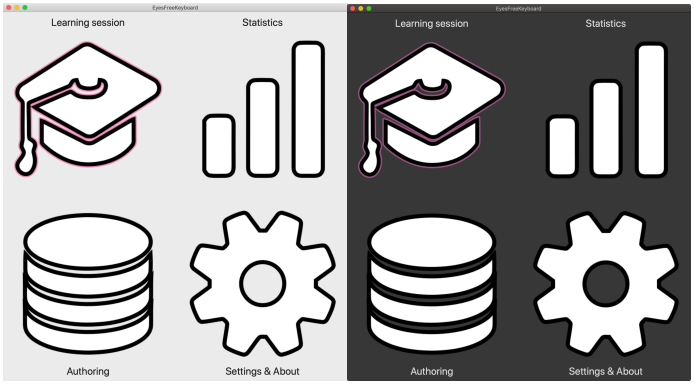


Abbildung R-IV2e-1: Abbildung des Hauptmenüs (unterstützt helles und dunkles Thema)

Klickt man auf den Knopf für die Lehrsitzung starten so kann man sich für ein Thema entscheiden. Die Themen bei dem Tippsystem sind statisch und basieren auf den verschiedenen Bereichen des Tippsystems in denen verschiedene Tasten und Tastenkombinationen enthalten sind. Die Bereiche sind anfangs sequenziell in ihrem Schwierigkeitsgrad, wechseln dann aber zu einem freieren System mit mehreren Themen pro Schwierigkeitsgrad. Das System schlägt einen nur Themen vor, die passend zu der Erfahrungsstufe des Profils sind. Nach einem gewissen Erfahrungsstufe kann man zwischen Einzeleingabe von Tastenkombinationen zu einer Serieneingabe wechseln. Die Serieneingabe ist für Wörter oder andere Reihenfolgen bei denen mehrere Symbole gleichzeitig angezeigt werden. Entweder sind das die Funktionstasten oder die schreibbaren Tasten. Die Themenwahl kann dabei konkret sein, wie man die Symbole auf den Zahlentasten tippt und merkt.

Nach der Wahl des Themas wird das informative Lehrmaterial präsentiert. Da im Prototypen das informative Lehrmaterial aus Text besteht ist das Oberflächendesign für diesen Teil des Lehrprozesses simpel gehalten. Der Text wird in Kapitel eingeteilt und jedes Kapitel hat eine Überschrift und einen Inhaltstext. Jedes Kapitel wird für sich angezeigt und durch einen Knopf lässt sich vorwärts von einem Kapitel zum nächsten springen, siehe dazu dazu Abbildung R-IV2e-2. Jedes Kapitel ist ein eigener Schritt der Erklärung des informativen Lehrmaterials und hat einen anderen Inhalt: Von der Einleitungen, Historie, Erklärungen des Tippsystems, Tipps und so weiter.

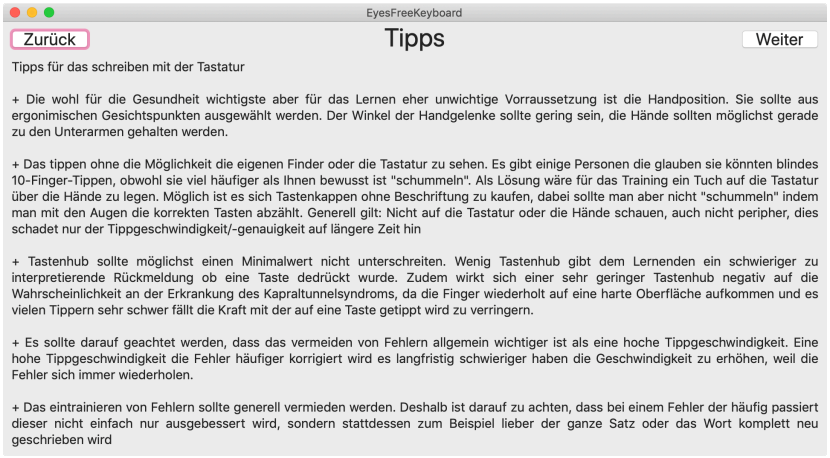


Abbildung R-IV2e-2: Beispiel des informativen Lehrmaterials

Ist man mit dem informativen Lehrmaterial fertig, folgt direkt danach das interaktive Lehrmaterial. Hier war das Oberflächendesign weitaus fordernder, siehe dazu Abbildung R-IV2e-3. Das Programm kann nicht nur die geschriebenen Symbole, sondern alle Tasten als Eingabe nutzen. Trainieren wie man zum Beispiel blind die Funktionstasten wie Escape oder F1 mit oder ohne Modifikatoren drückt ist möglich. Der Lernende ist unter Zeitdruck und hat wenig Zeit sich die Benutzeroberfläche anzuschauen. In der wird zentral ein oder mehrere Zeichen eingegeben, die als nächstes eingegeben werden sollen. Die Anzeige von einer Lehrsitzung mit Einzeleingabe zeigt immer nur ein Zeichen, weil sonst die Messung der Reaktionsgeschwindigkeit ungenau wird. Der Lernende entwickelt die Fähigkeit, schon den nächsten Buchstaben zu lesen während dabei eine vorherige Taste gedrückt wird. Für den Fall, diese Fähigkeit mit einzubeziehen, werden auch Lehrsitzungen für Serieneingaben gegeben. Für das genauere Testen der Fähigkeiten des Tippens von bestimmten Tasten ist die Einzelanzeige besser geeignet. Rechts und links davon die Markierung welche Finger dafür jeweils genommen werden sollen und im gleichen Abstand die Unterstützung der Anzeige der Tastatur. Der Kern ist dabei die Tastatur und Hände die angezeigt werden und farblich belegt werden. Dabei gibt es verschiedene Unterstützungsstufen, um den Schwierigkeitsgrad steigern zu können. Bei wenig Fertigkeiten wird die Tastatur und die Hände angezeigt. Beide zeigen farblich welche Finger genutzt werden und wo auf der Tastatur die gewünschte Taste sich sowohl für Modifikatoren als auch für Nicht-Modifikatortasten befindet, dies wird grün hervorgehoben. Wenn man die Taste ohne die Modifikatoren drückt werden die benötigten Modifikatorentasten in gelb hervorgehoben. Falsche Eingaben werden in rot hervorgehoben. Herausragende Werte in der Statistik werden bei positiver Verschiebung in grün und negativer in rot als Textfarbe hervorgehoben. Dies ist deshalb etwas trickreich, da sich die Modifikatoren und die Hand abhängig von der gedrückten Taste verändert. Die Unterstützung wird in mehreren Stufen anhand des Profils in Bezug zum Thema und dem Schwierigkeitsgrad reguliert. Folgende Stufen sind vorhanden.

1. Tastatur mit Beschriftung mit Markierung der Tasten und der Anzeige der Finger
2. Tastatur mit Beschriftung mit Markierung der Tasten
3. Tastatur mit Beschriftung mit Markierung der Tasten
4. Nur die eine Taste in richtiger Position
5. Tastatur mit Beschriftung
6. Keine Unterstützung

Informationen über die gedrückte Taste und die Modifikatoren stehen links oben im Fenster, darin kann man sehen welchen Keycode, Symbol und Funktion oder Ausgabe die Taste hat. Darunter das gleiche für die Modifikatoren. Als Modifikatoren werden die vom Betriebssystem unterstützten angezeigt: Befehlstaste, Optionstaste, Steuerungstaste und

Capslock. Informationen über die Statistiken der im interaktiven Teil der aktuellen Lehrsitzung stehen oben rechts.

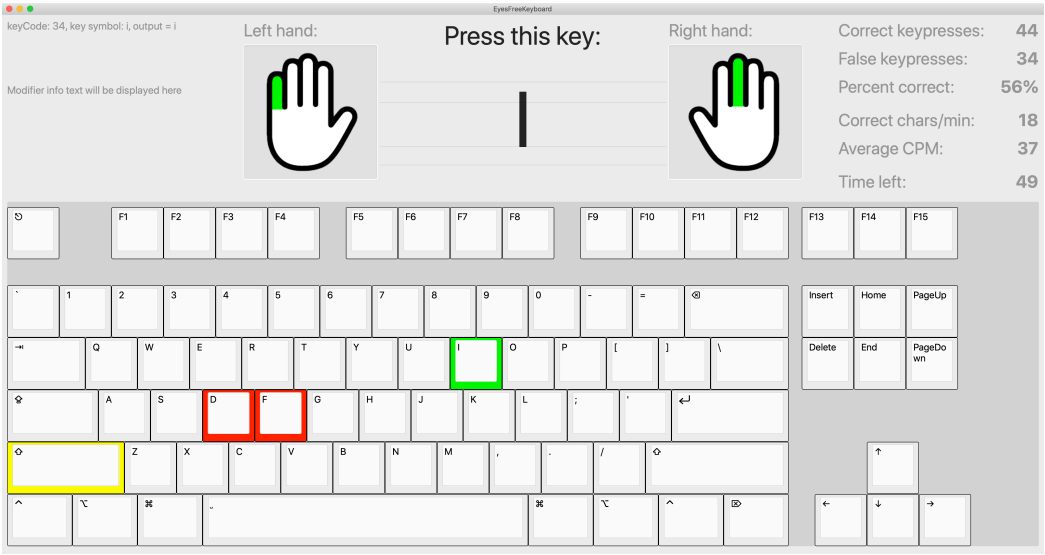


Abbildung R-IV2e-3: Beispiel des interaktives Lehrmaterials

Am Ende der Lehrsitzung, nach dem interaktiven Lehrmaterial, wird die Evaluation der Lehrsitzung ausgeführt. Dort werden Single-Choice-Fragen, welche die subjektive Meinung des Lernenden behandeln, gestellt, siehe Abbildung R-IV2e-4. Es gibt mehr sinnvolle subjektive Fragen als gestellt werden, doch nur wenige sollten gestellt werden damit der Lernende die Fragen nicht ignoriert oder schlimmer noch, die Fragen absichtlich falsch beantwortet. Deshalb werden nur wenige Fragen gestellt die innerhalb kurzer Zeit beantwortet werden können und die für den Prototyp relevant sind, weil sie umsetzbar sind. Die Eingaben werden gespeichert und können später für die Profilanalyse verwendet werden. Welche Fragen man stellt und was für Rückschlüsse man dabei zieht, kann ganz unterschiedlich sein. Es wurde bei diesem Prototypen versucht den Lernenden nicht mehr Stress als nötig auszusetzen. Dabei zielt die ersten drei Fragen darauf ab, abzuschätzen ob das Profil trotz der vorliegenden Daten sich mehr in eine Richtung die dem Lernenden besser gefällt lehnen sollte. Wurde zum Beispiel eine Lehrsitzung erfolgreich gemeistert, würde das Profil objektiv davon ausgehen, dass mit dem nächsten Material fortgefahren werden kann. Fühlt sich der Lernende allerdings überfordert obwohl er objektiv in der Lage ist die Aufgaben zu lösen, so sollte man ihm trotzdem erlauben eine für ihn schon bekanntes Lehrmaterial zu lösen um ein Gefühl von Sicherheit zu vermitteln. Im Gegensatz dazu gibt es Lernende, die sich langweilen, wenn sie nicht gefordert werden. Bei der zweiten und dritten Frage ist es auch reine Präferenz, ob ein Lernender lieber ausführlich oder kurz in ein Thema eingeführt und/oder getestet werden will. Die letzte Frage versucht ein Gesamturteil über die Lehrsitzung zu schaffen, falls eine Lehrsitzung generell schlecht bewertet wird, könnte man in der Lehrsitzung versuchen irgendwas stark zu ändern und sehen ob es besser wird. Über mehrere Lernenden ist die letzte Frage eine sehr gute allgemeine Einschätzung der Qualität einer Lehrsitzung, die sich bei konstanter schlechter Qualität vermeiden lässt. Solche Rückschlüsse sind für den Aufbau von Lehrwerkzeugen in der Domäne äußerst nützlich.

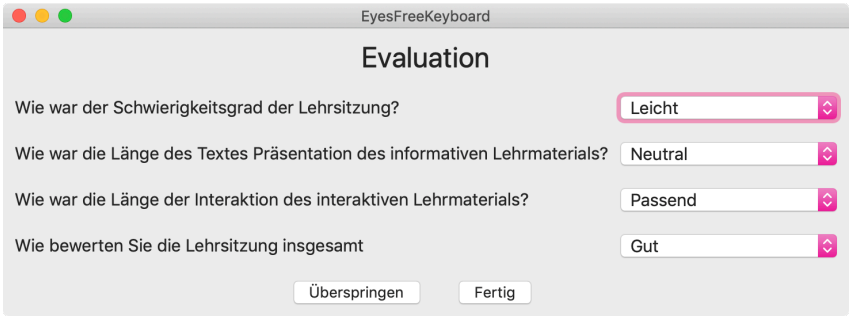


Abbildung R-IV2e-4: Beispiel der Evaluation

Vom Hauptmenü aus kann kann man auch auf die Statistiken zugreifen. Sie bildet die Statistikszene ab und zeigt die Zusammenfassungen als auch das Profilfeedback. Die Zusammenfassungen werden in mehreren Seiten in Diagrammen dargestellt. Jedes Diagramm fasst eine gesammelte Information über den Lernenden visuell zusammen, siehe Abbildung R-IV2e-5. Man kann jedes Diagramm als pixelbasiertes Bild exportieren und im PNG-Dateiformat abspeichern.

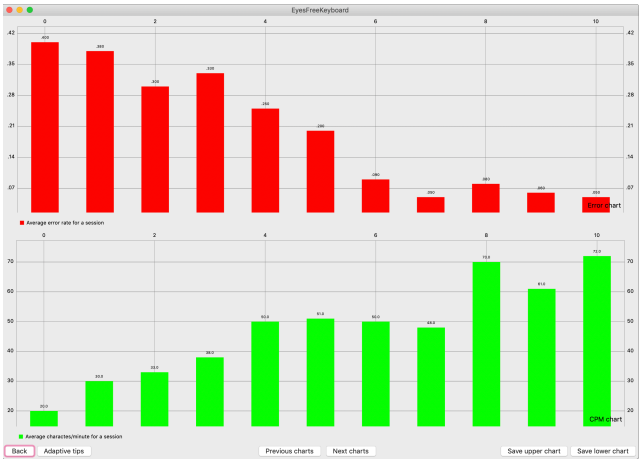


Abbildung R-IV2e-5: Beispiel der Statistik

Der letzte Menüpunkt sind die Einstellungen und Informationen über die Applikation. Darin lassen sich kleinere Einstellungen über die Applikation ändern die keinen wesentlichen Einfluss

Es wurde sich absichtlich nicht zu sehr auf die Benutzeroberfläche oder die Datenbankstruktur fokussiert, weil dies nicht Teil des zu überprüfenden Softwareframeworks ist. Es wurde mit diesen äußeren Schichten hauptsächlich überprüft ob es generell ohne Schwierigkeiten verwendet werden kann ohne dabei außergewöhnliches Design zu verwenden.

V. ZUSAMMENFASSUNG UND DISKUSSION

V.1. Zusammenfassung

V.1.a. Skizzierung der Arbeitsschritte

Der Forschungsbereich des ITS ist sehr groß und es gibt eine sehr gute Aussicht auf hohe Effektivität. Der Wunsch nach besseren ITS ist da und gut funktionierende ITS werden gebraucht. Mit dieser Doktorarbeit wurde ein Problem in den vorhandenen ITS-Systemen erkannt: Die Ad-Hoc-Entwicklungen von modernen ITS-Systemen zur Lösung des immer wieder gleichen Problems. Zuerst wurde beschrieben warum heutzutage ITS relevant sind, danach das Problem beschrieben, spezifiziert und erklärt warum es sinnvoll ist das Problem zu lösen. Von Anfang an war das Ziel, die Analyse und Verbesserung der allgemeinen ITS-Softwarearchitektur gesetzt. Eine allgemeingültige klar definierte Beschreibung der ITSs sollte das Ergebnis sein. Damit es konkreter wird, welche Ziele erreicht werden sollen, wurden diese beschrieben und in überprüfbare Thesen transformiert. Der Nachvollziehbarkeit halber wurde eine Liste angelegt und immer aktuell gehalten mit allen Werkzeugen, Ressourcen und Quellen die verwendet wurden. Somit kann immer in einem Kapitel nachgelesen werden, mit welchen Mitteln und Daten gearbeitet wurde.

Bevor angefangen werden konnte die aktuelle Lage von ITS zu analysieren, wurden die relevanten theoretischen Grundlagen des Lehrens und Lernens beschrieben die dafür relevant sind. Dazu zählt besonders die Definition von Fachbegriffen, die in ITS-Entwicklungen benutzt werden. Es wurden erst die einfacheren Wörter definiert und basierend darauf immer kompliziertere Definitionen wie die Lerntheorien und Lernmodellen erklärt. So entstand ein Wortschatz mit einer einheitlichen Definition auf denen die notwendigen Begriffe der ITS aufbauen kann. Somit folgte die Einführung in das Thema der Lehrsysteme und deren Entwicklung hin zu den ITSs, durch die historische Aufzählungen vom Thema der Digitalisierung über die Geschichte der Lernsysteme hin zu der Definition eines klassischen ITS und seiner Komponenten. Erst jetzt konnte man ein ITS hinreichend genau definieren. Die Ursprungsidee des ITS war eine Weiterentwicklung eines Expertensystems, doch in dieser Arbeit wurde analysiert ob überhaupt das Ziel erreicht wird: Ein ITS soll ein Lehrer sein. Deshalb wurde untersucht welche die für Software relevanten Fähigkeiten eines Lehrers sind und ob diese übernommen werden. So konnte die erste Aufteilung in die Bestandteile von ITS und deren vorkommen in vorhandenen ITSs definiert werden. Im Zuge der Analyse wurden die allgemeinen Nachteile, die vorhandenen Softwareframeworks und die Anwendung in der Konsumwirtschaft und in der Forschung recherchiert.

Es war klar, dass eine neue Softwarearchitektur her musste und nach einem Rundblick wurde spezieller das Thema des Softwareengineering im Zusammenhang mit mit ITS-Architekturen fokussiert. Zum Vergleich und dem Erkenntnisgewinn wurden ausführlich die existierenden Softwarearchitekturen aufgeführt und statistisch nach wichtigen Fragestellungen in Bezug zu einer allgemein gültigen Architektur ausgewertet. Entscheidende Erkenntnisse über die Grundlagen von ITS im Softwareengineering wurden getroffen, darunter fällt, welche allgemeine Softwarearchitektur sich eignet und welche Statistiken es über die vorhandenen Softwarearchitekturen es gibt. Dadurch konnte eine qualifizierte Meinung über die ITS-Softwarearchitekturen gemacht werden. Es bestätigte sich nicht nur der Verdacht des Problems, sondern es wurden viele Probleme mit Hinblick auf den bisherigen Standard detailliert beschrieben. Zwar wurden Verbesserungsvorschläge gemacht um die klassische Softwarearchitektur weiterzuentwickeln, doch war der Anspruch der Thesen größer als eine reine Verbesserung umsetzen könnte. Selbst die Weiterentwickelte klassische Softwarearchitektur im bisher fehlenden UML-Modell genügte nicht den Anforderungen. Alle Voruntersuchungen kamen zu dem Schluss, dass eine kleine Weiterentwicklung der klassischen Softwarearchitektur nicht ausreicht. Durch die Erkenntnisse und dem

erlangten Wissen konnte eine Beschreibung gefunden werden, welche Grundlagen sich für ein ITS-Modell und ITS im allgemeinen besonders eignen.

Um eine Neuentwicklung zu gestalten wurden jetzt erst eigene Vorüberlegungen angestellt, diesmal aus einem neuen Blickwinkel als sonst üblich, erst den Prozess zu entwickeln und funktional aus dem Prozess die Architektur anzugehen. Bevor der Prozess entwickelt werden konnte, mussten weitere Analysen zu vorhandenen ITS eingeholt werden. Es wurde analysiert und definiert wie der ITS-Entwicklungsprozess aufgebaut ist und welche Personengruppierungen daran beteiligt sind, mit jeweils ihren eigenen Ansichten, Vorstellungen und Wünschen. Es wurde ein Problem erkannt, dass die Verschiedenheit der zugrundeliegenden Systeme zu schwer vergleichbaren Daten führt die nur schwer Neuentwicklungen beeinflussen. Der erste Schritt für die Vereinheitlichung war die Definition der Forschungsbereiche [2019a GRAF VON MALOTKY and MARTENS] im Forschungsbereich ITS. Es wurden die drei Forschungsbereiche definiert und sich auf den ersten Forschungsbereich festgelegt. Als nächstes wurden die Paradigmen von ITSs recherchiert und erklärt. Die Gemeinsamkeit wurde erkannt, dass sie mit dem Lehrmaterial anders umgehen als Lehrer und deshalb wurde das Lehrmaterial passend für ITS aufgeteilt und spezifiziert. Dann wurden alle Szenarien von ITS-Aufbauten erforscht und detailliert beschrieben. Das ganze Wissen über die verschiedensten ITSs, deren Aufbau und Anwendung ist notwendig um zu Wissen welche Erwartungen und Definitionen für ITSs explizit und implizit vorhanden sind und damit die wichtigsten Anforderungen zu entwickeln. Um nicht zukünftige ITSs außen vor zu lassen, wurden durch die Szenarien alle Kombinationen ausgelotet die praktisch möglich sind. Die wichtigsten Anwendungsfälle die vorkommen wurden nochmal speziell abstrahiert aufgeführt.

Die nächste Anforderung für ein generisches Modell eines automatisierten Lehrers war der Vergleich mit einem echten Lehrer. Es wurden die klassischen Komponenten auf ihre Anwendungsfähigkeit innerhalb der Funktionsanalyse eines Lehrers gemacht. Der Lehrer ist mit der höchsten 1:1 Lehreffizienz das Vorbild. Es war deutlich dass der menschliche Lehrer über viel Wissen verfügt, welches genauer differenziert werden kann um dann formalisiert auch in einem ITS eingesetzt zu werden. Der Zustand eines ITS wurde beschrieben und aufgeteilt in den Bereich, welche persistiert werden muss und den Bereich der nur zeitweise benötigt wird. Eine Fähigkeit die der Lehrer ad-hoc beherrscht ist das automatische Generieren von Lehrmaterial. Diese Fähigkeit ist zwar in ITSs nicht umgesetzt, stellt aber einer der zukünftigen Ziele dar, die erreicht werden sollten. Es wurde klar definiert welches Wissen fehlt, um solch eine Funktion zu gewährleisten und wie man dieses Wissen speichern müsste. Eine Definition über die Wissensschichten wurde erarbeitet sowie eine erste Einteilung des persistierten Wissens.

Der Prozess ist der erste Baustein der Ergebnisse und kann die ganze Vorarbeit zusammenführen. Es wurde absichtlich nicht mit der Struktur, sondern mit dem Ablauf der Interaktionen mit einem ITS begonnen, weil damit vermieden wird, dass wie in der klassischen Architektur nur die Datenbanken definiert werden. Den Prozess gibt es in vier Abstraktionsebenen und definiert damit die Interaktionsabläufe mit einem ITS. Der Prozess wurde in UML formalisiert [2017b GRAF VON MALOTKY, NICOLAY and MARTENS]. Der Prozess wurde auch mittels eines endlichen Automaten modelliert, welches sehr unübersichtlich wurde, deshalb wurde er als Petri-Netz modelliert und als interaktive Variante bereitgestellt. Aus diesem Prozess wurden die Grundlagen gewählt die für ein ITS geeignet sind und aufgeschrieben.

Auf Basis dieses Prozesses und der ganzen Vorarbeit konnten die funktionellen Anforderungen an eine Softwarearchitektur langsam erarbeitet werden. Beginnend mit der Anforderung und der Zielsetzung eines Konzepts bis zu einem formales UML-Modell. Es wurde die 5-Schichten-Architektur als Basis gewählt und für die entsprechenden Bedürfnisse angepasst. Jede Schicht wurde definiert, mit den anderen Schichten wenn nötig verknüpft und dann formalisiert. Für jede der inneren Schichten wurde der Vorgang wie-

derholt. Funktionsanforderungen stellen, Analyse der Umsetzung, bestmögliche Gruppierung des Funktionsangebots, Definition der Einzelteile, Verknüpfung mit den anderen Komponenten und formalisieren. Dies wurde dann nochmals für die dann entstandenen Komponenten gemacht. Es erreichte ein Detailgrad bei dem eine noch feinere Granularität der allgemeinen Anwendung der Architektur geschadet hätte. Wenn man eine Architektur wachsen lässt ohne sich vorher zu informieren, was andere schon gemacht haben und von klein beginnt und langsam größer wird, dann ist es wahrscheinlich, dass man bestimmte Fehler so wiederholt wie es andere vor einem gemacht haben. Eine größere komplexe Architektur zu nehmen, die mit guter Recherche entwickelt wurde, die sich auf genau das zu entstehende Endsystem fokussiert vermeidet dies. Der Lehrprozess wurde beispielhaft an der Architektur nochmals angewendet um zu zeigen wie die Funktionsanforderungen überprüft werden.

Mit der Softwarearchitektur werden nicht alle Fragen beantwortet, ein großer Teil der Arbeit steckt in der Implementierung. Es ist allerdings ein großer Teil schon vordefiniert und zwar der Teil der abstrahiert werden kann mit Sicht auf die Funktionsweise, den Elementen aus dem ein ITS besteht und wie diese zusammenarbeiten. Um zu überprüfen, ob die Implementierung Fragen oder Probleme aufwirft die bisher unentdeckt blieben ist es notwendig nicht nur in der Theorie zu bleiben. Bis auf wenige Anpassungen konnte die UML-Formalisierung problemlos angewendet werden. Das liegt unter anderem auch daran, dass durch die Formalisierung in das UML-Modell welches ein Werkzeug des Softwareengineering ist, viele Probleme einer eventuellen Programmierung schon erkannt und behoben wurden bevor sie überhaupt wirklich programmiert wurden. Um auch einen Implementierung zu bieten die leicht an Neuentwicklungen verwendet werden kann, wurde nicht gleich an einem Prototypen entwickelt, statt dessen wurde für den Einsatz der Architektur ein Softwareframework entwickelt. Dieses setzt die Architektur um und bietet gleichzeitig eine weitere Sicht auf diese. Der Aufbau und die Umsetzung der Architektur in ein Framework wurde beschrieben. Das Softwareframework wurde in zwei Projekte aufgeteilt, eines für größere und eines für kleinere Entwicklungen. Für den Einsatz wurde beschrieben wie man in einem Projekt das Softwareframework einbindet und verwendet.

Es konnte mittels des Softwareframeworks ein Prototyp entwickelt werden, der adaptiv an den Lernenden Tastaturschreiben mit 10 Fingern lehrt. Der Aufbau die Zielsetzung und die Umsetzung der Anforderungen an solch ein Programm wurden beschrieben. Die Anforderungen des Prototypen wurden von dem Softwareframework erfüllt und alle konkreteren Schritte wurden implementiert.

V.1.b. Beantwortung der Thesen

Nachdem diese Arbeit zu einem Resultat gekommen ist können die Thesen, welche am Anfang dieser Arbeit aufgestellt wurden überprüft werden. Folgend sind alle Thesen aufgeführt und es wird einzeln darauf eingegangen.

These 1

Es lässt sich ein einheitlicher Prozess der Interaktion eines ITS mit einem Lernenden entwickeln.

Bisher gab es keinen eigenen einheitlichen Lehrprozess für ITSs. Vom IEEE wurde versucht eine Architektur mit einem Prozess zu versehen, welches damit keine Softwarearchitektur darstellt. Dieser Standard wurde zurückgezogen. Der Gewinn eines Prozesses wurde erkannt aber falsch angegangen. Ein Prozess würde eine Vereinheitlichung der Vorgehensweisen von ITSs definieren. Mit einem einmal durchgesetzten Standard wären ITSs vergleichbarer, intuitiver für Lernende, leichter zu verstehen, entwickeln und warten, weil der Umgang und die offenen Optionen bereits bekannt sind. Die Best Practice ist mit

einer Vereinheitlichung besser, besonders zwischen den einzelnen Fachbereichen kommunizierbar.

In dieser Arbeit wurde der ITS-Lehrprozess für die vorhandenen ITSs definiert. Dazu wurden bekannte ITSs und deren implizite Vorgehensweisen, die Ideen der Verwendung von gefundenen Softwarearchitekturen und der Lehrer mit seinen didaktischen Vorgehensweisen analysiert. Die vorgestellte Lösung inkludiert damit nicht nur Prozesse von existierenden ITSs, sondern hat zusätzlich die Fälle analysiert, in der ein ITS eingesetzt werden könnte aber bisher nur theoretisch wird. Es ist damit bestens auch für zukünftige Herausforderungen gewappnet, in der die technologische und KI-Entwicklung weiter voranschreitet. Daraus entstand der in Kapitel II.4.c. vorgestellte Prozess, welcher unabhängig von einer Architektur ist und damit auch in Bereichen verwendet werden kann, in der eine Softwarearchitektur unnötig ist.

Es ist damit die These bestätigt, dass es möglich ist einen einheitlichen Ablauf zu Entwickeln die aus Sicht des ITS, sowohl die Initialisierung als auch den Umgang mit den Lernenden während des Lehrvorgangs mit den Interaktionswegen zu definieren.

These 2

Der Prozess aus These 1 kann in verschiedene, unabhängige Detailstufen aufgeteilt werden.

Der Prozess hilft der Vereinheitlichung und dem Verständnis von diesem wichtigen Ablauf. Doch eine Vereinheitlichung hat nicht nur das Ziel angewendet zu werden, sondern auch als Kommunikationsmittel zu dienen. Beide Teile benötigen für ihren speziellen Zweck eine bestimmte Abstraktionstiefe. Ein Modell in mehreren Abstraktionstiefen anzubieten erhöht ihren Nutzen und die Wahrscheinlichkeit auch Außerhalb der Programmierung genutzt zu werden. Deshalb wurde der aus These 1 vorgestellte Prozess auch in mehreren Abstraktionsebenen definiert (Kapitel II.4.c.). So hat eine unspezifischere Umsetzung oder Anwendung über den Prozess genauso ihre Definition wie eine detaillierte. Einmal wurde der Gesamtprozess in seine wichtigen Schritte im Groben geteilt. Dort hat man einen guten Überblick und kann die Anforderungen und den Aufwand abschätzen. Dann wurde jeder Bereich einzeln beschrieben. Dabei entstanden in jedem Bereich des Gesamtprozesses eigene Teile. Der jeweilige Subprozess hat Eingänge und Ausgänge und ließe sich mit Beibehaltung von diesen austauschen, ohne den Gesamtprozess zunichte zu machen. Das Ziel der These nicht nur eine grobe Einteilung zu bekommen, sondern diese auch noch in feineren detaillierten Schritten zu beschreiben war einheitlich für die ITS umsetzbar. Damit wurde diese These erfolgreich erfüllt.

These 3

Der Prozess aus These 1 lässt sich formal spezifizieren.

Wie in Kapitel II.3.b. zu sehen gibt es in der Entwicklung mehrere Beteiligte aus den unterschiedlichsten Bereichen die für eine gutes ITS benötigt werden. Um diese nicht auszuschließen wurde letztendlich eine visuelle Formalisierung vorgezogen. Die Umsetzung in UML als einer der größten Modellierungssprachen erlaubt einen Fülle an Ressourcen zum Verständnis bis hin zu Automatisierungssoftware. Der Anwendungsspielraum ist damit schon von Beginn an hoch. UML ist sehr ausführlich formalisiert und es wurden nur Elemente aus dieser Sprache genommen. So gibt es eine eindeutige Bedeutung der Elemente und eine Verifizierung wird vereinfacht. In Kapitel II.4.c. wurde der Prozess im UML Aktivitätsdiagramm modelliert.

These 4

Es lässt sich eine einheitliche Architektur für intelligente Lehr-/Lernsysteme entwickeln.

Die Abstraktionsebene von bisherigen Softwarearchitekturen ist fast immer unpassend. Entweder sind es Architekturen die zu speziell sind und die nicht verallgemeinert werden können, oder sie sind für ein ITS mit den heute zu Verfügung stehenden Werkzeugen ohne viel eigene Modellierungsarbeit kaum nützlich. Für die verallgemeinerten Architekturen ist das Problem ihr Alter. Um für so einen langen Zeitraum gültig zu bleiben ohne große Veränderungen anzunehmen, während es eine rasende Entwicklung mit Computern/ Smartphones/Tablets gab, mussten sie zwangsläufig zu grob sein. Die Anleitung für eine Implementierung ist unklar und eine genauere Spezifikation fehlt. Die klassische Architektur ist schon Jahrzehnte alt und nicht an die Anforderungen der heutigen Systeme angepasst. Diese Architekturen spiegeln hauptsächlich die Sicht der groben Strukturierung der Datenbanken wieder und vernachlässigen die Funktionalität und größtenteils auch das Design. Die aktuelle klassische ITS-Architektur wird häufig so stark unterschiedlich aufgefasst, dass sie nicht mehr kompatibel untereinander sind und damit auch deren Ergebnisse sehr ungleich aufgebaut sind. Der große sichtbare Vorteil den Lernende mit einem ITS in der vorgestellten Softwarearchitektur haben, sind die folgenden Funktionen: Vorschlagen von Lehrthemen, Anpassen des Lehrmaterials, Rückmelden zu gegebenen freien Eingaben und angepasste Hilfe. Jedes von den Funktionen sind Funktionskomponente die wiederum auf anderen Funktionskomponenten auch in der ITS-Kernfunktionalität beruhen. Der Ansatz die Funktionalität als gleichwertigen Teil zur Oberfläche und zur Datenbank zu betrachten und diese in ITS-Kernfunktionalität und Programmfunktionalität aufzuteilen war neu. Neuere Architekturen sind entweder spezielle Architekturen aus einem existierenden Prototypen oder haben die gleichen Probleme, siehe dazu Kapitel III.1.d. zu den Problemen von Softwarearchitekturen. Eine ausführliche Analyse in den Anfangskapiteln brachte die wichtigsten Anforderungen für das Konzept (Kapitel II.5.) und für die Software (Kapitel III.1.f.). Aufbauend darauf wurde sowohl der de-facto Standard erweitert (Kapitel III.1.e) als auch eine komplett neue Architektur erzeugt, welche diese Probleme behandelt Kapitel III.2.h.. Neue Anforderungen wie die automatische Generierung von Lehrmaterial (Kapitel II.3.k.) und die zentrale Wissensdatenbank wurden definiert und diese wurden auch in den anderen Schichten wieder aufgenommen. Bekannte Anforderungen wie die Autorenfunktionalität wurden durch ihre Einbettung in explizite Komponenten klarer definiert und deren Wichtigkeit gestärkt. Der klare Ansatz des Softwareengineering aus III.1. führte zu einer Struktur, die ganzheitlich die ITS widerspiegelt als auch den mit den Qualitätsansprüchen von heute umsetzbar ist.

These 5

Die Architektur aus These 4 kann in verschiedene, unabhängige Detailstufen aufgeteilt werden.

Es war besonders wichtig bei einem komplexen System wie ein ITS die Integration von Experten außerhalb der Softwareentwicklung anzusprechen. Dies konnte nur mit verschiedenen Abstraktionsebenen erreicht werden. Beginnend mit einer Black Box hin zu einer detaillierten allgemeinen Softwarearchitektur für Entwickler ist dies gelungen. Mit mehr Komponenten wird es einfacher, die Softwarearchitektur zu implementieren, weil sie detaillierter und genauer ist und weniger Freiheiten für Fehler lässt. Leider erhöht es für abstraktere Kommunikation oder für ein grobes Verständnis des Konzepts auch so sehr die Komplexität, dass es für andere Aufgaben unbrauchbar wird. Anstatt unterschiedliche und inkompatible Architekturen aus den Abstraktionsebenen zu erschaffen, wurde eine einheitliche Softwarearchitektur in mehreren Detailstufen definiert. Die These wurde damit bestätigt, dass es möglich ist, die Details in vier Abstraktionsebenen so zu verallgemei-

nern, dass es in der Kommunikation und Vermittlung von höheren Konzepten bis hin zur Unterstützung bei der direkten Implementierung mit detaillierten Beschreibungen eingesetzt werden kann. Im Kapitel III.2. wurden die 4 einzelnen Abstraktionsebenen jeweils in dem dazu passenden Detailgrad (Kapitel III.2.c. bis Kapitel III.2.h.) erläutert.

These 6

Die Architektur aus These 4 lässt sich formal spezifizieren.

Es wurden verschiedene Wege ausprobiert, wie eine formale Spezifikation umgesetzt werden kann ohne dabei alle Experten anderer Fachbereiche den Einstieg zu schwer zu machen. Letztendlich wurde sich für eine visuelle Variante entschieden, weil selbst Personen, die wenig Vorwissen in formalen Beschreibungen in der Informatik haben, inhaltlich grob verstehen, was es zu bedeuten hat. Dabei wurde einer der weit verbreitetsten Modellierungsstandards genommen: UML (Kapitel III.2.). Diese Modellierungssprache ist ausgereift und erfüllt seinen Zweck sehr gut und kann einheitlich mit der Prozessformalisierung aus These 3 bleiben.

These 7

Der Prozess aus These 3 und die Architektur aus These 6 sind miteinander kompatibel.

Auch wenn die Teile jeder für sich haltbar und nutzbar sein sollten, so ist eine Synergie der beiden Definitionen vorteilhaft. Es wurde erfolgreich in dieser Arbeit geschafft Architektur und Prozess unabhängig voneinander verwenden zu können. Sowohl der Prozess als auch die Softwarearchitektur wurden nach dem gleichen Prinzip aufgebaut und wurden beide in UML definiert. Inhaltlich beschreiben sie zwei Bereiche des selben Ziels: Die Entwicklung eines ITS. Der zuerst formalisierte Prozess aus These 3 verdeutlichte die Wichtigkeit des Prozesses, sodass in der Gestaltung der Architektur aus These 6 Rücksicht auf einen von außen definierten Prozess zu nehmen ohne sich auf einen zu spezialisieren (Kapitel III.2.i.). Die dafür geschaffene Komponente der Architektur wurde speziell so definiert, dass der Prozess frei gestaltet werden kann. Deshalb erlauben die Berechtigungen der Komponente durch Verknüpfungen einen größeren Spielraum als der aus These 3 aufgezeigte Prozess. Es wurde erfolgreich überprüft, ob der Prozess aus These 3 ohne Probleme mit dieser Architektur verwendet werden kann. Beide decken grundsätzlich alle notwendigen Funktionalitäten von ITSs ab, so wie sie in dieser Arbeit konzeptionell erarbeitet wurden. Für eine Erleichterung der Zusammenarbeit der beiden Modelle in einem Konzept wurde in den Ergebnissen darauf geachtet, auch stilistisch klar und konsistent zu sein, um Details der These erfolgreich umzusetzen.

These 8

Die Architektur aus These 6 kann in einem objektorientierten Softwareframework implementiert werden.

Mit einem Softwareframework kann der Entwickler den Tranformationsschritt von UML in eine Programmiersprache überspringen und direkt mit dem Programmieren beginnen. Dabei kümmert sich das Softwareframework um die Einhaltung der Regeln der Softwarearchitektur. Es bietet einen Einstieg und konkrete Programmierkonstrukte um die Modellelemente direkt einsetzen zu können. Softwareframeworks für ITS gibt es nur eine Hand voll, doch keines von ihnen besaß als Grundlage eine so einheitliche und gleichzeitig detaillierte Softwarearchitektur, welche gleichzeitig frei von Anforderungen einer Plattform oder Prototypen entstand. Die bisherigen Softwareframeworks haben entweder wenig Beschränkungen und es ist dadurch unabsichtlich schnell möglich einer falsche Implementation zu folgen oder sie haben die Form von einer Sammlung von Softwarewerkzeugen die immer nur eine Art an ITSs erzeugen kann. Es wurde die Softwarearchitektur mit-

tels eines Softwareframeworks ohne Abstriche in Swift implementiert (Kapitel IV.1.). Die Herangehensweise bleibt wie bei der Architektur und setzt sie bestmöglich um. Dies ließ sich sehr gut mit die Fähigkeiten der gewählten Programmiersprache abbilden. Das Softwareframework wurde in zwei Varianten erstellt, um die Last der Komplexität bei kleineren Projekten geringer zu halten gab es eine flache Implementierung und eine in der alle Abstraktionsebenen jeweils ihre eigene Abbildung haben. In beiden Varianten kann der Nutzer frei entscheiden welche Komponenten er selbst implementiert. Die Anzahl der Eigenentwicklungen entscheidet über den Funktionsumfang des ITS. Dabei hilft auch die Beschreibung der Gruppierungen was ITS-Kernfunktionalität und Programmfunktionalität ist. Der Entwickler kann sich also auf die Umsetzung der gewollten Komponenten fokussieren und überlässt den Rest dem Framework.

These 9

Ein ITS kann mit dem Softwareframeworks aus These 8 implementiert werden. Dieses hält die Architekturvorgaben aus These 6 mittels des Softwareframeworks und gleichzeitig ist dessen Prozess festgelegt auf den in These 3 definierten Prozess.

Eine Entwicklung eines ITS-Prototypen unter vollen Verwendung des Softwareframeworks ergab, dass sowohl der Prozess als auch die Softwarearchitektur in ihrer Praktikabilität die Ansprüche mehr als erfüllte (Kapitel IV.2.). Die Verwendung des Softwareframeworks zeigte, dass eine konkrete Umsetzung mit dessen Hilfe schnell umsetzbar war und keine zusätzlichen Konzepte in der vorgestellten Abstraktionsebenen benötigte. Eine Einhaltung der vorhandenen Definitionen und Verknüpfungen führte zu einer klaren Aufteilung der Verantwortlichkeiten und Anforderungen, sodass die Konkretisierung der äußeren Schichten wie beabsichtigt die domänenspezifische einzige Modellierungshürde darstellte. Der Prototyp erfüllt seine Lehraufgabe und kann als gelungene Erzeugung eines ITS die These als korrekt belegen.

V.1.c. Weitere Resultate

Das Resultat inkludiert nicht nur die Softwarearchitektur und den Lehrprozess, sondern auch die Definitionen und die Vorarbeit dahin. Eigene neue Definitionen und Ergebnisse sind in dieser Arbeit vielzählig und können auch ohne die großen Resultate in der Forschung und Anwendung wiederverwendet werden. Diese Ergebnisse werden folgend aufgezählt, dabei außen vor gelassen werden Irrwege und ungenutzte Nebenprodukte, diese werden in Kapitel V.2.b. extra aufgeführt.

- Kapitel „ITSs in der Forschung“: Es gibt eine Statistik der Auswertung der ITS-Softwarearchitekturen. Diese wurde in der größten und anerkanntesten Konferenz für ITSs öffentlich verteidigt [2019b GRAF VON MALOTKY and MARTENS].
- Kapitel „Probleme der gefundenen ITS-Softwarearchitekturen“: Es gibt eine Auflistung der Probleme mit aktuellen ITS-Softwarearchitekturen.
- Kapitel „Vorschläge für die klassische Softwarearchitektur“: Es gibt eine Weiterentwicklung der klassischen Softwarearchitektur mit formaler visueller Definition in UML.
- Kapitel „Wahl der Konzeptgrundlage“: Es gibt eine feste Herangehensweise für ITS als Auswahl von Lerntheorien Lernmodellen.
- Kapitel „Einordnung von ITSs in Lehr-/Lernsystemen“: Es gibt eine Taxonomie für die Einteilung der Gruppierung und Benennung von Lernsoftware.
- Kapitel „ITS-Entwicklungsprozess“: Es gibt eine Definition des ITS-Entwicklungsprozesses mit der Aufzählung aller Beteiligter. Dies wurde öffentlich verteidigt [2019a GRAF VON MALOTKY and MARTENS].

- Kapitel „ITS-Forschungsbereiche“: Es gibt eine Definition über die Forschungsbereiche bei ITS. Dies wurde öffentlich verteidigt [2019a GRAF VON MALOTKY and MARTENS].
- Kapitel „Aufteilung des Lehrmaterials“: Es gibt eine Aufteilung des Lehrmaterials und die Definition der Gruppen davon.
- Kapitel „ITS-Anwendungsszenarien“: Es gibt die Definition der Rollen, Aktionen und Ortsbeziehungen von ITS die für die Lehre wichtig sind.
- Kapitel „ITS-Anwendungsszenarien“: Es gibt die Definition aller Szenarien für die ein ITS eingesetzt werden kann.
- Kapitel „ITS-Anwendungsfälle“: Es gibt die Definition über Hauptanwendungsfälle von ITS.
- Kapitel „Menschlicher Lehrer versus ITS“: Es gibt eine Definition der klassischen Komponenten in Bezug zu einem Lehrer.
- Kapitel „Zustand eines ITS“: Es gibt die Definition über den Aufbau eines Zustands in dem sich ein ITS befinden kann.
- Kapitel „Automatische Generierung von Lehrmaterial“: Es gibt eine Definition von der automatischen Generierung von Lehrmaterial. Wie das Lehrmaterial aufgebaut werden muss und welche zusätzlichen Daten man dafür braucht. Dies wurde öffentlich verteidigt [2017a GRAF VON MALOTKY, NICOLAY and MARTENS].
- Anhang: Es gibt eine Auflistung aller gefundenen Softwarearchitekturen mit dazugehöriger Visualisierung.
- Anhang: Es gibt eine Auflistung aller Namen für die klassischen Komponenten.

V.2. Diskussion

In den über 30 Jahren der ITS-Entwicklung wurde kein detaillierter Standard angegangen. Der Blickwinkel aus einem allgemein gültigen Prozess heraus die Funktionalitäten zu entdecken und sich nicht nur auf die Benutzeroberfläche oder die Datenbanken zu konzentrieren ist einzigartig. Im Rahmen der Digitalisierung durfte hier keine weitere Ad-Hoc-Entwicklung entstehen, sondern ein für diesen Forschungsbereich wichtige Prozessbeschreibung mit allumfassender Softwarearchitektur. Dieser neue Ansatz dieser Arbeit ist deshalb relevant, weil er als Pre-Standard dienen kann, um eine Grundlage für vergleichbarere Software zu haben. Es erleichtert die Kommunikation mit einem Standarisierten Wortschatz. Es lässt einfacher Komponenten austauschen, wenn die gleiche Programmiersprache verwendet wird. Es lässt leichter neue Systeme begreifen, weil sie den gleichen Grundaufbau haben. Es lässt leichter Funktionsweisen und Algorithmen übertragen, weil die Aufteilung und Aufgabenverteilung grob gleich bleibt. Es lässt leichter Daten austauschen, weil die Aufteilung der Datenbank klar definiert ist.

Man erkennt dass viele der Entscheidungen die innerhalb dieser Arbeit während des Schreibens getroffen wurden, diese Arbeit lässt einen teilhaben an der Entwicklung der Ergebnisse. Diese Entscheidungen wurden begründet gewählt. Zum Verständnis des nächsten Schritts war es wichtig für die Begründung von zentralen Aspekten der Arbeit nicht erst in der Diskussion erwähnt zu werden. So konnte ein roter Faden innerhalb der Arbeit etabliert werden, an dem der Leser sich entlang orientieren kann. Mir war es auf diese Art möglich auf erzählerische Weise den Leser teilhaben zu lassen am Fortschritt der Doktorarbeit durch eine gemeinsame Suche zur Lösung des ursprünglichen Problems. Es entstanden trotzdem viele Fragen und Wahlmöglichkeiten der Vorgehensweise, welche unabhängig vom roten Faden betrachtet werden konnten. Die wichtigsten Fragen, welche während der Entwicklung aufgetreten sind werden in den kommenden Kapiteln beantwortet.

V.2.a. Qualität der Hauptresultate

Die Hauptergebnisse sind der formalisierte Lehrprozess, die formalisierte Softwarearchitektur, das Softwareframework und der Prototyp.

ITS-Lehrprozess

Das erste Hauptresultat ist der formalisierte Prozess der Lehrablaufs von Lernenden und ITSs. Ein unabhängiger einheitlicher Lehrprozess für ITS gab es bisher noch nicht und ist ein großer Schritt vorwärts für die Forschung von ITSs. Der Prozess besitzt mehrere Schichten und kann damit sowohl im Groben als auch in feineren Abläufen verwendet werden. Die Definition des Prozesses wird in der weit verbreiteten Modellierungssprache UML umgesetzt, die mit ihren sehr genauen Definition der Einzelelemente eine sehr gute Grundlage bildet. Es wurde das Resultat sowohl visuell und formal in UML in den verschiedenen Abstraktionsebenen definiert, als auch mittels Prosa die Anwendung der Einzelschritte beschrieben. UML ist in der Softwareentwicklung sehr verbreitet und bietet den Programmierern einen Leitfaden der Zeitlichen Interaktionen und Aufgaben in einem klar definierten Rahmen. Er dient zusätzlich auch im Speziellen zusätzlich als ein gutes Kommunikationsmittel zwischen den einzelnen Experten außerhalb der Programmierer. Die dann nicht nur sachlich schwer zusammenhängende Probleme hintereinander besprechen sondern dies in Kontext setzen können und Verknüpfungen der Probleme zueinander erkennen können. Sie haben damit ein Artefakt über dass nicht nur gesprochen wird, sondern auf dass gezeigt und von ihnen an die jeweilige Domäne klarer definiert werden kann. Dieses Ergebnis bietet einen Überblick über alle Teile die ein ITS beherrschen muss und in welcher Reihenfolge sie auftreten, sodass sich auch Personen die sich

mit ITS nicht so auskennen sehr schnell die Aufgabe und die Anforderungen ein Bild machen können. Ein Prozess bietet einen leichteren Einstieg in die Thematik als eine Softwarearchitektur, weil sich dort ein Anwendungsfall leichter durchspielen lässt. Er ist auch eine sehr gute Vorlage für das Verständnis der Verwendung der Softwarearchitektur in zeitlicher Abfolge. Mit ihm ist man im Stande die Elemente der Funktionalitäten einer Softwarearchitektur in Bezug zu setzen und damit zu erkennen was davon in welchem Teil notwendig ist.

Dieses Ergebnis bietet eine einzigartige Möglichkeit für die Standarisierung eines der Kernfragen der ITS-Entwicklung des Handlungsablaufs, des Ablaufs aus der internen Sicht des ITS oder aus Sicht des Lernenden. Eine ganzheitliche Beschreibung als Standard würde die Definition was ein ITS ist klarer machen und weitere Forschung daran vereinfachen. Durch die Formalisierung ist dieser eindeutig und kann gegebenenfalls als Verifikationsmittel verwendet werden. Für den Nachweis der Machbarkeit wurde der Prozess in einem Prototypen erfolgreich eingesetzt und die zweckmäßige Verwendung der Einzelschritte überprüft. Das Resultat des ITS-Prozess wurde öffentlich verteidigt mit peer review in einem Paper [2017b GRAF VON MALOTKY, NICOLAY and MARTENS].

Der Prozess und die Architektur wurden so entwickelt, dass der Stil sich sehr ähnlich ist. Dieser Vorteil wurde aber nicht zu sehr ausgeschöpft, denn dadurch dass es unabhängig von der Softwarearchitektur ist, werden auch keine Komponenten der Architektur in dem Prozess referenziert die in den einzelnen Schritten verwendet werden sollten. Auch wenn eine gemeinschaftliche Lösung angestrebt wurde, war es wichtiger dass der Prozess losgelöst von der Architektur verwendet werden kann. Eine Entwicklungshilfe für den Prozess wurde nicht so stark umgesetzt wie für die Softwarearchitektur. Es gibt kein Softwareframework, welches automatisch für die Implementierung und Einhaltung des Ablaufs sorgt. Der verallgemeinerte Prozess kann für spezielle Domänen angepasst werden und damit noch genauer definiert werden. Dies wurde in dieser Arbeit nicht gemacht.

Softwarearchitektur

Das zweite Ergebnis ist die formalisierte Softwarearchitektur. Die unabhängige Entwicklung der Softwarearchitektur und erst danach das Softwareframework und dann den Prototypen zu beachten war eine gute Entscheidung für die Qualität der Architektur. Die Ideen kamen nicht von einer Eigenentwicklung, sondern erst nach gründlicher Recherche von vorhandenen ITS-Architekturen und ITSs. Bei der Wahl welche Komponenten in der Architektur vorhanden sein sollen, geht es nicht darum ob die Funktionalität gebraucht wird, sondern ob die Komponente in der richtigen Abstraktionsebene sind oder ob zwei Komponenten zusammengelegt oder aufgeteilt werden sollten. Wenn die Funktion einer Komponente sehr wichtig und zugleich deutlich trennbar ist, auch wenn das zur Umstellung anderer Komponenten führt, wird eine neue Komponente eingefügt oder eine bestehende aufgeteilt. Wenn zwei Komponenten stark voneinander abhängig sind und jede für sich genommen keine Daseinsberechtigung hat oder wenn die Funktionalitäten sich sehr ähneln und beide nur einen geringen Umfang haben, werden sie zusammengelegt. Irgendwann passiert es, dass eine Zusammenlegung und eine Aufteilung beide richtig erscheinen, dann hat man zwei Abstraktionsebnen der gleichen Sache. Entscheidend für die gefundenen Komponenten ist dass sie nach der passenden Gruppierung und Abstraktion gewählt wurden und nicht nach Wichtigkeit sortiert wurden. Alle Komponenten die vorkommen sollten wichtig sein, sein könnte man sie auch weglassen. Für was sie wichtig sind, ob für das Programm im Allgemeinen oder für die Umsetzung der Funktionalität des ITS wird in der Einteilung von ITS-Kernfunktionalität und Programmfunktionalität berücksichtigt.

Es gibt Ziele die bisher in der Forschung vernachlässigt wurden und das zählt im speziellen für die vorhandenen Softwarearchitekturen. Eines dieser Ziele ist das Autorensystem. Eine Autorensystem wird häufig vernachlässigt, weil der Aufwand zu hoch ist und

der Nutzen für Entwickler kaum vorhanden ist. Wozu ein System erweitern um eine Dateneingabe, wenn man direkt die Daten und Verhaltensweisen in der Datenbank oder im Code manipulieren kann. Eine Regel in Code aufzuschreiben ist simpel. Eine Regel konfigurierbar zu machen, bedeutet es muss irgendwo die Konfiguration gespeichert werden. Gespeichert wird dann in einer domänenspezifischen Sprache (DSL), dazu muss man es in beide Richtungen konvertieren können. Da die DSL meist nicht intuitiv zu verstehen ist, führt das zu einer eigenen Benutzeroberfläche innerhalb des ITS oder eines externen Werkzeugs. Eine Oberfläche für die eigenen Datenbankcode zu schreiben der alle Fälle abdeckt ist ebenso ein großer Aufwand. Jemand der sich aber nicht mit dem System auskennst, dafür aber umso mehr mit der zu unterrichtenden Materie. Dieser Aufwand lohnt sich langfristig bei der Erweiterung der Wissensbasis von Domänenexperten ohne Programmierkenntnisse, deshalb wurde er direkt in die Softwarearchitektur integriert. Es werden auch die anderen definierten Rollen des Betreuers und die Fernunterstützung von Lehrern berücksichtigt. Betreuer spielen bei der Betrachtung des Unterschieds der Motivation zwischen einem menschlichen Lehrer zu einem ITS eine überaus wichtige Rolle. Autoritäten die keine Menschen sind, sind sehr schwierig umzusetzen und das technische Wissen um alleine mit einem Computer umzugehen oder bei scheinbaren Fehlern aufzugeben darf in anderen Gesellschaften nicht unterschätzt werden. Auch wenn in westlichen Ländern der Umgang mit Smartphones und Computern intuitiv erscheint, so kann die Anwesenheit eines Menschen viel bewirken. Lehrer als Spitze der Lehreffizienz sollen zwar in bestimmten Bereichen entlastet werden, deren Hilfe sollte auch in einem kurzfristigen Rahmen betrachtet werden. In unvorhergesehenen Situationen könnten menschliche Lehrer bei ITS genau die Lücken füllen, wenn Maschinen versagen ohne dabei voll eingespannt zu werden. Somit wäre es vielleicht möglich einen Lehrer für mehr Lernende gleichzeitig einzusetzen, als er alleine im Stande wäre.

Ein weiteres Ziel ist die Nutzung eines ITS als Werkzeug für die Forschung. Ein Anwendungszweck von ITSs der nicht in seinem vollen Potenzial ausgeschöpft wird. Nicht den Lehrer versuchen zu ersetzen und menschliches Verhalten nachzuahmen, sondern die Vorteile einer Maschine nutzen. Durch die exakte Replizierbarkeit des Verhaltens beliebiger Versionen und Lehrstrategien statistisch sicherer auswerten und vergleichen zu können, während das Verhalten nicht nur nur einmal existiert, sondern beliebig häufig kopiert und parallel eingesetzt werden kann. Es erlaubt Forschern Rückschlüsse mit mehr Sicherheit zu treffen, wenn die Datenmengen einheitlicher, mit gleichen Bedingungen und größer sind. Diese Ergebnisse wiederum könnten wieder zurück zu menschlichen Lehrern getragen werden, um deren Lehre zu optimieren.

Explizite Einbettung eines austauschbaren Prozesses schützt vor einer impliziten harten Definition innerhalb einer Implementierung. Die Architektur ist auf eine Zusammenarbeit mit einem Prozess ausgelegt ohne sich dabei an den vorgestellten Prozess festzulegen. Der Prozess selbst ist dabei konzentriert auf eine Komponente und nicht in im ganzen Softwareframework verteilt. Dabei bleibt sie eine echte Softwarearchitektur und modelliert keinen Prozess intern. Die Architektur selbst verwendet den weit verbreiteten Grundaufbau in der Softwareengineering: Die Schichtenarchitektur und die Komponentenarchitektur. Sie kann leicht Teile der Funktionalität, des Aussehens oder der Speicherung durch den Komponentenbasierten Aufbau ändern, bei dem jede Komponente mit einer anderen Implementierung ausgetauscht werden kann, weil die Schnittstellen und die Funktionalitäten gleich bleiben.

Die Architektur besitzt eine formale und visuelle Repräsentation, die auf einem weit verbreiteten Standard der Modellierung basiert, der Unified Modelling Language (UML). Die gesamte Architektur versucht den gleichen Stil wie der Prozess einzusetzen, dass betrifft das Aussehen, die Beschriftung und auch den gleichen Modellierungsstandard wie der Prozess. Es ist somit einfach beide zusammen zu verstehen oder das eine zu verstehen wenn man das andere kennt. ITSs sind als Werkzeug für die Sammlung von Forschungs-

daten perfekt geeignet, weil sie unendlich häufig exakt das selbe Verhalten immer wieder reproduzieren können und dann parallel mit vielen einzelnen Lernenden zeitgleich. Die Sammlung der Daten können zusammengeführt und hochgeladen werden. Genau dieser Anwendungsfall wird auch in der Softwarearchitektur berücksichtigt.

Lernende mit sozialen Ängsten oder Kommunikationsproblemen, zum Beispiel im Extremfall Autisten oder Menschen mit Asperger-Syndrom, oder Personen die besonders unter Stress in sozialen Situationen stecken, haben mehr Probleme mit einem menschlichen Lehrer als durchschnittlich. Sie könnten von einer Kommunikation über und mit einem System profitieren. Auch Personengruppen bei denen es unhöflich ist dem Lehrer zu widersprechen (kommt häufiger in asiatischen Regionen vor) könnten mit einem Lehrsystem offener und ehrlicher kommunizieren [2016 KATADA]. Der Grundaufbau der Softwarearchitektur wurde öffentlich verteidigt mit peer review in einem Papier veröffentlicht [2016 GRAF VON MALOTKY and MARTENS].

Der einseitige Blick auf eine hohe Qualität sorgte auch für eine fehlende Abwärtskompatibilität mit der klassischen Softwarearchitektur. Eine Umwandlung der bisherigen ITS ist also aufwendig. Im Vergleich zu heutigen ITSs und der klassischen Architektur ist die komplette hier resultierende Architektur mit der Anforderung alle Komponenten und Funktionalitäten zu implementieren viel anspruchsvoller. Eine Reduzierung auf nur die Abstraktionsebene 2 ist aber vergleichbar.

Softwareframework

Das dritte Hauptresultat ist das Softwareframework. Es hat viele Vorteile sowohl im Einsatz als auch gute Entscheidungen in der Entwicklung. Es hilft beim schnellen Einstieg und Umsetzung eines Projektes. Es zeigt die Ziele und Bereiche die konkret notwendig sind und ist für einen Programmierer auch eine Art der Dokumentation der Architektur. Denn die Dateien sind so benannt und organisiert, wie es auch die Architektur ist. Somit ist es nicht nur einfach von der Architektur auf das Framework zu wechseln und mit der Implementierung zu beginnen, sondern auch umgekehrt kann erst mit dem Framework begonnen werden und dann durch die gleichen Aufbau schnell alle Elemente wiederzuerkennen. Das Framework funktioniert ohne weitere Programmierung und erlaubt eine schrittweisen Ausbau des ITS, wenn mehr und mehr gebrauchte Komponenten mit der eigenen Implementierung ersetzt werden. Man hat die Wahl, ob man selbst die Anforderungen durch annehmen eines Protokolls von Grund auf selbst baut oder mit einer rudimentären Implementierung startet und das zu jedem Zeitpunkt. Die zwei Versionen abhängig von der Größe des zu erwartenden Projekts ermöglichen einen flexibleren Einsatz. Selbst wenn das Projekt wächst kann man durch die fast identische Implementation mit wenig Arbeit auf die komplexere Version upgraden. Auch hier ist das Softwareframework unabhängig von dem Prototypen entwickelt. Das Softwareframework ist sowohl als open source Quellcode vorhanden, als auch als eine Datei mit der FRAMEWORK-Endung. Seine praktische Anwendung wurde in einem Prototypen getestet und bestätigt.

Nachteile des Softwareframeworks könnten durch die verwendete Programmiersprache Swift entstehen. Gerade durch diese Sprache wurde die gute Umsetzung der Architektur gewährleistet, sie ist noch relativ neu. Sie befindet sich deshalb noch in einer Phase, in der sie mehr Veränderungen durchlebt, als Jahrzehnte bestehende Sprachen. Veränderungen verbessern zwar die Programmiersprache, allerdings sind sie ab und zu noch so gravierend, dass der Quellcode der älteren Version von dem Übersetzer der neuen Swift-Version als inkorrekt angesehen wird. Trotz der bisher unübertroffenen Beliebtheit für eine neue Programmiersprache, ist die Verbreitung auf alle Plattformen nicht garantiert. Jede Wahl der Konkretisierung wird unweigerlich andere ausschließen, so auch hier.

Prototyp

Das letzte Hauptresultat ist ein Programm, welches ein blindes 10-Finger-Tippsystem lehrt und dabei auf den Lernenden eingeht. Auch wenn es sehr schwierig ist als einzelne Person ein ausgefeiltes ITS zu entwickeln, so wurde durch eine clevere Wahl der Domäne mit Erfolg ein ITS mit abgrenzbaren Bereich geschaffen. Er implementiert erfolgreich ein ITS auf Basis der bisherigen Vorarbeit. Der Prototyp passt sich an den Lernenden an, indem das informative und interaktive Lehrmaterial weniger oder mehr Hilfestellungen beinhaltet. Dazu wurde das Domänenwissen gestückelt und es wird immer nur das Wissen präsentiert welches zum Profil passt. Das informative Lehrmaterial ist damit gut aufgestellt sich der ausgefeilten Analyse des Lernendenwissens anzupassen. Es wurden sich viele Gedanken um einen guten Aufbau des Lehrmaterials gemacht. Anstatt dies sequenziell zu ordnen, wurde dem Domänenwissen Annotationen hinzugefügt, um sie abhängig vom Lernendenwissen herauszusuchen. Dies gilt auch für das interaktive Lehrmaterial. Hierbei handelt es sich um die möglichen Tastenkombinationen und deren Reihen mit Schwierigkeitsgrad und Verknüpfung zu didaktisch Vorwissen und gut darauf folgendes Lehrmaterial. Es werden die Aufgaben dadurch auch individuell ausgewählt. Dabei gibt es für die Anpassung des interaktiven Lehrmaterials eine Unterstützung des Lernenden und das wurde in mehreren Stufen integriert.Es gibt eine Evaluation der Lehrsitzung um die Einschätzung des Lernenden zu bekommen. Es wurde ein Autorenwerkzeug implementiert um eine einfache Benutzeroberfläche für Autoren erzeugt. Es wurde auch die Statistik implementiert, die dem Nutzer einen Einblick in die eigenen Leistungen gibt. Dort gibt es auch Empfehlungen vom ITS durch die Analyse des Profils. Der Stil des Prototypen wurde dabei absichtlich so gestaltet, dass das Projekt, die Programmiersprache und der Codestil gleich oder sehr nah zu dem des Softwareframeworks ist. Damit kann nachdem man sich in das Softwareframework eingearbeitet hat, den Prototypen als ein Beispiel verwenden um zu verstehen wie man es implementieren kann. Es ist damit nicht nur ein Machbarkeitsnachweis, sondern kann auch als Teil der Dokumentation gesehen werden. Der Prototyp hilft beim Verständnis wie man das Softwareframework anwendet.

Bei dem Erfolg den der Prototyp für das Konzept und die Architektur bringt, zeigt sich, dass das Endprodukt rein von der Zielgruppe an vorhandenen Nutzern beschränkt ist. Der Lehrinhalt hat Beschränkungen, die sich auf die am meist verwendeten Standards stützen aber nicht alle Spezialfälle abdecken. Für die Unterstützung des interaktiven Lehrmaterial gibt es kein frei einstellbares Hardwarelayout. Der Prototyp respektiert nur die Standardgröße und -verteilung der realen Tasten auf den Tastaturen. Es könnte, besonders bei Laptoptastaturen, ergonomischen oder selbstgebauten Tastaturen dazu kommen, dass die Anzeige nicht mit der Realität übereinstimmt. Auch das Softwarelayout ist festgelegt. Der Prototyp unterstützt keine selbstgefertigten Tastaturbelegungen oder Abweichungen vom Standard. Auch das Tippsystem ist nicht das Einzige für ein Standardlayout. Der Prototyp vertritt nur ein Tippsystem. Anderen Tippsysteme bei denen die Handzuordnung im speziellen für die Taste „b“ und „6“ mit dem Finger der gegenüberliegenden Hand getippt werden, werden nicht berücksichtigt. Als native App auf einer Plattform wird der Nutzerkreis weiter eingeschränkt. Eine Webapplikation wäre für den Nutzerkreis vielleicht besser geeignet, hätte aber andere Nachteile. Dass der Prototyp nur in einer Sprache angeboten wird, wird die Nutzerschaft auch begrenzen. Weitere Lokalisierungen wären vorteilhaft. Ein ITS als Test für das gesamte Softwareframework (damit auch die darin verwirklichte Architektur) welches nicht nur die komplette Kernfunktionalität verwendet, sondern auch jede einzelne Komponente der Programmfunktionalität des ITS-Softwareframeworks wäre gut geeignet als Testversuch in einem größeren Rahmen. Fernunterstützung von Lehrern und Betreuer sind Funktionalitäten die sehr aufwendig in der Programmierung sind.

Es fand keine Evaluation des Prototypen in Anwendung statt. Es wäre möglich die Anforderungen für ein ITS zu erfüllen und dabei trotzdem eine schlechte Lehrsoftware zu sein. Das Ziel des Prototypen war hingegen auch nicht hauptsächlich eine gute Lehrsoftware zu sein, sondern den Nachweis der Machbarkeit und einfachen Umsetzung der Softwarearchitektur mittels des Softwareframeworks in einem real existierenden ITS. Es wurde darauf geachtet, dass dieser Prototyp den Ansprüchen an guter Lehrsoftware gerecht wird und im kleinen Rahmen auch Rückmeldungen von Testern erhalten.

V.2.b. Irrwege und ungenutzte Nebenprodukte

Das größte Fehlkonzept in das Arbeit gesteckt wurde war der Versuch den ITS-Lehrprozess durch einen endlichen Automaten darzustellen um eine mathematische Formale Version zu bekommen. Der gesamte Prozess wurde in einen endlichen Automaten transformiert, dennoch war es nur sehr schwer ersichtlich, wie die einzelnen Aktivitäten sich untereinander beeinflussen. Der Wunsch, dass das Konzept insbesondere weniger nur von Softwareexperten verwendet wird, konnte damit nicht verwirklicht werden. Eine Verifikation machte diese Vorgehensweise attraktiv, die Umsetzung führte zu einer Explosion des Zustandsraumes und machte es damit insgesamt sehr schwer kompliziert und unleserlich. Es war das falsche Modell dafür. Eine leichte Verständlichkeit wäre vielleicht für Mathematiker gegeben in der Formelschreibweise gegeben, andere Personen würden es mit dieser mathematischen Schreibweise schwerer begreifen.

Es gab eine weitere Transformation des Lehrprozesses in ein anderes formales Modell: Es wurde als Petrinetz definiert. Diese Formalisierung wurde nicht nur auf Papier, sondern in einem Softwarewerkzeug umgesetzt, welches eine nachträgliche Simulation des Netzes erlaubt. Mit den CPN Tools kann man interaktiv die Marken setzen und so sehen wo man sich gerade befindet und wie das Netz darauf reagiert. Das funktioniert gerade bei parallel Ablaufenden Unterprozessen sehr gut. Das Werkzeug ist nicht sehr verbreitet und Petrinetze selbst nicht so weit verbreitet wie UML. Es wurde sich zugunsten der Einheitlichkeit mit der Softwarearchitektur auf UML festgelegt. Das UML-Modell war für außenstehende leichter verständlich und das Petrinetz verlor damit seinen Nutzen.

Es war die Idee vorhanden, dass es eine freie Wahl der Komponenten gibt und eine Auswahl an Komponenten als eine Konfiguration einem allgemeinen Anwendungszweck gleich kommt. Um die Konfigurationen und ihre Aufgaben vergleichen zu können wurde eine Tabelle mit jeder Kombination aus den Komponenten, die in einer ITS-Implementation verwenden werden kann, erstellt. Zu jeder Konfiguration wurden die möglichen Anwendungsfälle beschrieben. Die Hälfte der Konfigurationen waren dabei schon im Vorfeld als ungünstig aufgefallen und aussortiert. Es wurde untersucht welche Komponenten unumgänglich sind für ein ITS. Es war eine Idee diese Konfigurationen durch Parameter für einen Anwendungsfall nach ihrem Nutzen zu sortieren und eine beste Konfiguration vorzuschlagen. Der Gedanke wurde weiter geführt und es wurde ein Programm geschrieben, dass diese Konfigurationen auswählbar macht. Das Hilfsprogramm zur einfachen Erstellung von einem ITS-Projekt mit dem Grundgerüst an Klassen welche man für den Start braucht wurde erstellt. Mit dessen Hilfe ist es möglich einen schnellen Einstieg in die Entwicklung zu bekommen, indem die ersten drei Schritte automatisch ausgeführt werden. Man muss nur noch das Hilfsprogramm öffnen und die Funktionalitäten anwählen, welche man benötigt, siehe dazu Abbildung R-V2b-1. Man kann mit Klicken Häkchen setzen bei den Komponenten, hier eine Vorversion der Softwarearchitektur mit englischen Namen, bei dem Extensions die Programmfunktionalität und Features die ITS-Kernfunktionalität darstellt. Sobald man „Build“ anklickt wird das neue ITS-Projekt erstellt.

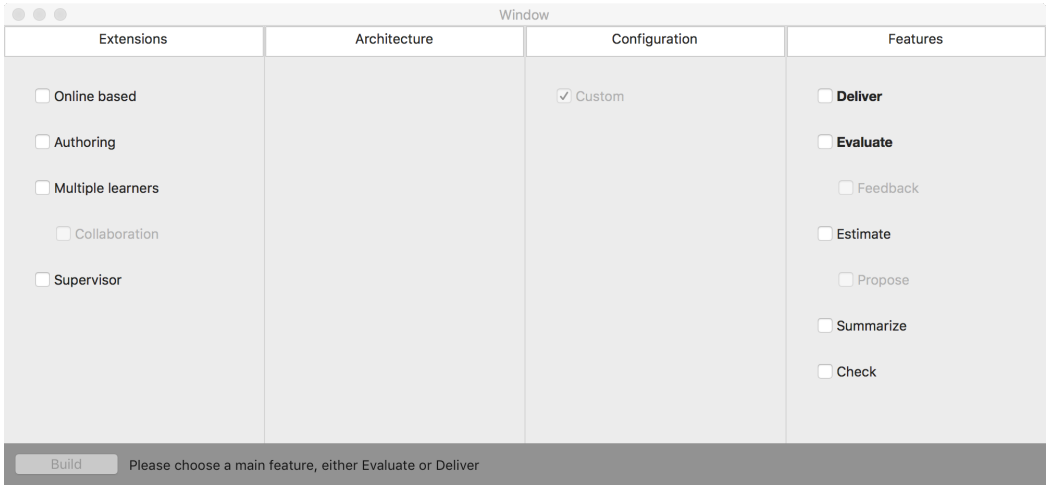


Abbildung R-V2b-1: Bildschirmfoto der ITS-Erstellungssoftware

Dabei würde jeweils eine Konfiguration ein Softwareframework aus vorhandenen Quelldateien zusammenstellen und dann ein Projekt erstellen und dieses einbinden. Das Hilfsprogramm wurde nicht fertig gestellt. Bei einer kleinen Anzahl an Komponenten war die Generierung eines neuen Frameworks trivial und es mussten nur wenige Kombinationsmöglichkeiten beachtet werden. Doch mit Weiterentwicklung der Softwarearchitektur und mehr Komponenten in der vierten Abstraktionsebene wurde die Komplexität hoch. Der Wartungsaufwand stieg, denn durch die Komplexität anhand der Variationsmöglichkeiten stieg auch die Fehlerquote. Eine neue Variante ohne Helferprogramm eines statischen Softwareframework konnte so gestaltet werden, dass eine Integration in eine vorhandenes Projekt sehr einfach wurde und damit der Nutzen der ITS-Erstellungssoftware gering war. Auch hatte der statische Ansatz den Vorteil, dass er eine einfachere Integration (zum Beispiel erlaubt es eine Stück-für-Stück-Integration) in bereits bestehenden Projekt funktioniert, während die Erstellungssoftware nur bei einem komplett neuen Projekt angewendet werden könnte. Der statische Ansatz erforderte auch eine statische Softwarearchitektur und so hatte es Auswirkungen auf die Softwarearchitektur. Es wurde sich entschieden, dass der Gewinn die Komplexität nicht überwiegt. Besonders bei Weiterentwicklung des Softwareframeworks wäre es unnötig anderen Programmierern erschwert, die sich in die Codebasis einarbeiten wollen. Die Idee eine Softwarearchitektur in Konfigurationen aufzuteilen mit jeweils ihrer eigenen Beschreibung half bei der Gruppierung und dem Verständnis bei einer Neuentwicklung einer Architektur. Mit dem neuen Ziel eines statischen Softwareframeworks und nach grober Fertigstellung der Softwarearchitektur war die Konfigurierbarkeit viel weniger von Nutzen. Es besteht sogar die Gefahr, dass eine Aufteilung der Softwarearchitekturen in Teilmengen bei einer Weiterentwicklung von nur einer der Untermengen zu einem auseinanderdriften führt. Für eine Vereinheitlichung wurde darauf verzichtet die Konfigurationen im Resultat zu verwenden. Geblieben ist die Idee, dass bestimmte Konfigurationen für die Forschung besonders interessant sein könnten, weil sie besonders selten oder häufig vorgekommen sind. Forschungslücken und Vorteile von vorhandenen ITS könnten unter Umständen erkannt werden. Diese wurde aber nicht weiter verfolgt.

Zusammenfassend haben sich die Ansätze nicht als gut genug erwiesen und sollten nur dann zukünftig in ähnlichen Ansätzen verwendet werden, wenn deren Nutzen in dem neuen Anwendungsfall vorher gründlich untersucht wurde. Es wurde nicht daran weitergearbeitet und weder die Konfigurationen sortiert, gruppiert noch genügend aussagekräftige Parameter gefunden, welche solch eine Klassifikation sicher erlaubt.

V.2.c. Entscheidungen im Konzept

Für die Entwicklung des Konzepts, bestehend aus den Modell von Prozess und Architektur und den daraus entstandenen Hauptergebnissen mussten viele Entscheidungen getroffen werden. Einige die besonders schwer zu treffen waren und wahrscheinlich damit für andere auch eine Hürde darstellen können wurden hier begründet.

Podcasts und Videos sind informative Lehrmaterialien auch wenn man innerhalb des Videos interaktiv navigieren kann. Das große Problem von Videos oder Podcasts ist meist deren Länge. Adaptive Hypermedia erzeugt aus mehreren Videos . Solange diese Videos kurz sind um nur passende abzuspielen ist solch eine Methode anwendbar um informatives Lehrmaterial zu erzeugen. Es entsteht dabei ein „Branching Program“ in dem Verzweigungen entweder manuell oder durch ein Test gewählt werden. Ein Vorgängermodell des ITS welches durch seine Einfachheit alle paar Jahre das Interesse der Lehrforscher gewinnt. Interaktives Lehrmaterial besteht aber nicht nur weil ab und zu eine oder mehrere Single/Multiple Choice Frage gestellt wird die das nächste Video festlegen. Die Anpassungsmöglichkeiten sind zu beschränkt, weil es viel zu aufwändig wäre so viele Videoschnipsel herzustellen. Das Problem zieht sich weiter, die Aufgaben und Antworten sind statisch und zu gering in der Anzahl, das System kann damit nicht auf die Probleme des Lernenden eingehen, sondern der Lernende kann sich nur in wenigen diskreten Optionen ausdrücken. Das Lernendenwissen wird reduziert auf wenige Optionen und damit zu grob als dass ein Profil viele Informationen analysieren kann. Die Auswertung des interaktiven Lehrmaterials ist die Hauptquelle der Einschätzung des Lernenden, weil dort das Wissen überprüft wird, was im informativen Lehrmaterial präsentiert wurde. Audio-/Video-Podcasts sind damit nur als informatives Lehrmaterial geeignet.

Ein Lerner kann Wissen nicht nur erlangen sondern auch vergessen. Das Lernendenwissens nicht nur in die positive sondern auch in die negative Richtung automatisch zu korrigieren wäre sehr nützlich. Dabei sollte sowohl die Zeit des letzten Lernens und die Häufigkeit mit der ein bestimmten Wissen gelernt wurde beim Modell des Erinnerungsvermögen bedacht werden. Ein einmalig erlerntes Wissen wird schneller vergessen als häufig gelerntes Wissen und desto länger es her ist umso wahrscheinlicher ist, dass das Wissen vergessen wurde. Es ist generell sehr ratsam einen Zeitstempel für die Einträge im Lernendenwissen zu haben, um eine Zeitliche Ordnung in den Interaktionen des Lernenden zu haben, dieser könnte auch hier verwendet werden. Dadurch lassen sich im Profil Rückschlüsse auf die Wahrscheinlichkeit des Vergessens von dem zeitlichen Verlauf der Lehrsitzungen in einem bestimmten Thema erstellen. Die Funktionalität des Vergessens wird nicht als extra funktionelle Komponente implementiert, da das Profil genau das abdecken kann. Die gespeicherten Rohdaten im Lernendenwissen dürfen nicht verändert werden um den aktuellen Wissensstand zu einem Zeitpunkt abzubilden. Mit dieser Historie des Wissensstands des Lernenden kann das was vergessen wurde berechnet werden.

Es gibt weitere spezielle Funktionen die das Lernendenwissen verzerren könnten. Unbeabsichtigte Fehleingaben vom Lernenden würden das Bild des Lernenden in die falsche Richtung beziehen und werden nicht immer vom Lernenden bemerkt. Selbst wenn es bemerkt wird, muss dann der Lernende noch immer die Bearbeitungsrechte bekommen für die Korrektur nur genau dieser Daten. Es gibt Ansätze die solche Fehleingaben entdecken [2006 EL-KECHAÏ and DESPRÉS], nachträglich kann der Nutzer entweder selbstständig entscheiden oder diese Eingaben werden direkt als Ausreißer angesehen und bei der Profilgenerierung ignoriert. Es kann eine Verzerrung aber auch mutwillig geschehen um dem System ein größeres Lernendenwissen vorzugaukeln. Wenn dies durch Eingriff auf die Datenbank führt ohne die Benutzeroberfläche zu verwenden, dann ist dies eine Manipulation und kein Fehler des Systems. Zu Prüfungen oder Prüfungsvorleistungen muss natürlich sicher gestellt werden, dass die Leistung auch wirklich eigenhändig erbracht wurde. Bei einem ITS ist dies durch eine Verschlüsselung der Daten selbst für

den lokalen Nutzer recht einfach möglich, erzeugt aber weitere Probleme in anderen Bereichen wie der Datenrettung. Es muss davon ausgegangen werden, dass Nutzer in unvorhergesehenen Fällen das System bewusst auch innerhalb der Benutzeroberfläche austricksen kann. Solch ein Schummeln wird auch „game the system“ genannt und kann mit bestimmten Techniken besser vermieden werden, zum Beispiel durch die Erkennung wenn ein Lernender Gamingverhalten zeigt [2006 WALONOSKI] oder direkt direkt an das Gamingverhalten adaptieren [2006 D BAKER, CORBETT and KOEDINGER]. Für ein ITS als Verifizierungswerkzeug in der Didaktik ist diese Eigenheit eine besonders wichtige Eigenschaft. Man kann sich auch leicht vorstellen, dass selbst ohne einen objektiven Profit, ein Schummeln langfristig nicht zuträglich für die Motivation des Lernenden ist.

Lernende die bereits Vorwissen haben könnten den Wunsch verspüren sofort in einen Fortgeschrittenen Bereich einzutauchen. Das kann zum Ausspielen des Systems führen. Der Grund für die Funktion ist es, dass ein Lernender keine Lust hat viele Lehrsitzungen mit Altbekannten durchzuarbeiten, er lernt nichts Neues und verliert die Motivation. Bei vielen Computer Assisted Language Learning Software ist dies leider der Standard [2019 VAWTER, GRAF VON MALOTKY and MARTENS] und die manuelle Eingabe des Vorwissens kann eine Lösung sein. Optimaler wäre hingegen, die Erkennung der überragenden Leistung des Lernenden und dann davon auszugehen, dass der Lernende Vorwissen besitzt. Das hat beide Vorteile, es verhindert eine Schummeln und es erlaubt Lernenden die in einem Gebiet überaus gut sind schneller als normal voranzuschreiten. Dabei wird sogleich das Problem der Selbstüberschätzung behoben. Dies tritt in einer Klasse von Lernenden auf, die das Thema schon behandelt haben und deshalb glauben sie könnten den Stoff schon. In Wirklichkeit besitzen sie weniger Wissen als von ihnen selbst eingeschätzt. Die Abschätzung was erwartetes Verhalten vom Lernenden wäre, kann auch weitergeführt werden um das Verhalten zu simulieren, welches einfache Testläufe ohne Zeitverschwendung durchlaufen kann [2006 BROWN and ESKENAZI].

Für das maschinelle Lernen und statistische Aussagen ist es notwendig eine hohe Anzahl von Daten (educational data) zu besitzen. Dies wird schwierig mit nur einem ITS. Wenn allerdings mehrere ITS auf dem gleichen Konzept aufgebaut sind und diese Daten vergleichbar sind, dann ist es auch möglich diese Daten in kleineren Mengen zu verwenden [2006 HÄMÄLÄINEN and VINNI]. Die Wiederverwendung und Analyse von großen Datenmengen waren einer wünschenswerten Vorraussetzungen für viele zukünftige Ergebnisse und Technologien. Es wurde sich zum Ziel gesetzt dies auch zu integrieren.

Bei algorithmischen Lösungsvorschläge wie SQL, Formeln oder einer Programmiersprache ist es einfacher die Adaptivität der Rückmeldung zu generieren. Dort lassen sich mehrere Testfälle schreiben um zu testen ob die Antwort des Lernenden korrekt ist. Für jeden Nichtbestandenen Testfall gibt es dann eine vorgefertigte Antwort die speziell Versucht das Problem zu beschreiben und zu erklären was der Lernende falsch gemacht hat. Wenn die Testfälle nach Schwierigkeit sortiert sind, ermöglicht es eine adaptive Antwort mit statischen Hilfstexten anhand einer Eingabe. Dies lässt sich auch an Lösungsvorschlägen des Lernenden ohne Eingabe machen, dann muss aber der Lösungsvorschlag vergleichbar mit einer vorhandenen Antwort sein die das ITS kennt, was viel schwieriger ist. Desto formeller die Lösungsvorschlag sein muss umso einfacher ist es für das ITS sie auszuwerten. Es wurde darauf geachtet, dass ITSs mit formeller Syntax und eindeutigen, begrenzten Wortschatz im Fokus stehen, auch wenn diese die Mehrheit der Systeme ausmachen. Denn die offeneren Systeme, welche zum Beispiel natürliche Sprache unterstützen werden zukünftig einen höheren Stellenwert ausmachen und darauf soll auch die Architektur vorbereitet sein.

Kommen wir zu den Entscheidungen welche generell für die Architektur wichtig waren. Von einem ITS wird generell mehr erwartet als von anderen Lehrsystemen, dennoch ist es möglich die Funktionen eines ITS auf ein Minimum zu reduzieren. Selbst dann muss eine Adaption vorherrschen und dafür werden viele Teile benötigt die in der Architektur aufge-

zeigt wurden. Um dies transparenter zu machen wurde die Programmfunktionalität eingeführt. Es gibt bestimmt auch andere Wege die Wichtigkeit von Komponenten im Bezug zur Erfüllung der ITS-Definition zu erfüllen. Es wurde eine gesucht und gefunden die auf einen Blick verständlich ist. Die Aufteilung der Persistenzschicht wurde nicht so gewählt sondern zugunsten der bekannten Aufteilung in der Abstraktionsebene größtenteils beibehalten. Es brauchte viele Iterationen um das Resultat zu kommen und die Zwischenentwicklungen der Softwarearchitekturen könnten in der Arbeit erwähnt werden. Die Erkenntnisse welche zum finalen Modell geführt haben wurden zusammengefasst und erwähnt aber keine verworfenen Fehlentwicklungen des Modells, welche wieder verworfen wurden, auch wenn sie wahrscheinlich notwendig waren um zum Resultat zu kommen. Auch wenn mit der klassischen Architektur begonnen wurde und sogar versucht wurde diese zu erweitern, wurde eine komplett neue Architektur begonnen. Der Einsatz des Konzepts in den unterschiedlichsten vorhandenen sowie den noch möglichen ITSs sollte abgedeckt sein. Anfangs wurden nur konkrete ITSs genommen und auf deren Anwendung überprüft. Das würde aber die zukünftigen ITSs außer Acht lassen und nur eine bestimmte Konfiguration der Möglichkeiten betrachten. Die möglichen ITSs, die entstehen können, sind zu viele. Deshalb wurde entschieden alle Kombinationen möglicher ITS mit deren Rollen, Verortungen und Aktionen als Modell zu erstellen und auch damit abzugleichen. Nach diesem Schritt wurde sich dafür entschieden, dass eine Generalisierung dieser Größenordnung Probleme mit der Konkretisierung haben wird. Deshalb wurde sich dagegen entschieden Datenbanken zu modellieren, sondern nur deren Vertreter des konzeptionellen Struktur der Daten in der Persistenzschicht. Die Versuche der Definition der Datenbanken mit Ausprägungen durch ER-Modelle wurde gestoppt.

Bei der Entwicklung der Architektur wurden die Verknüpfungen immer wichtiger. Es kam die Frage auf, ob die Richtung der Verknüpfungen in nur eine Richtung beschränkt werden soll. Wenn man die Schnittstellen so definiert, dass die eine Seite nur Funktionalität bereit stellt und die andere Seite diese Funktionalität nutzt und benötigt, dann hat dies starke Konsequenzen auf den Informationsfluss. Eintretende Ereignisse können dann nicht mehr in beide Richtungen getragen werden. Dies würde aber eine Schicht über die andere Stellen und damit eine Hierarchie erzeugen, dass heißt zum Beispiel wenn die Interaktionsschicht Aktionen des Nutzers bekommt, werden diese eine Kette Aufrufen in den darunterliegenden Schichten auslösen. Eine Helfenkomponente könnte dann allerdings nicht selbstständig entscheiden, aktiv einzugreifen ohne vorher einen Aufruf der oberen Schicht zu bekommen. Es gibt viele Beispiele bei denen solch eine Verknüpfung sinnvoll ist: Update der Datenbankänderungen, Netzwerkaktivität oder asynchrone Antworten. Vorteil wäre, dass die Architektur generell vereinfacht wird, weil es dann nicht mehr bidirektionale Referenzen gibt, die zyklisch sind und bei der eine Referenz schwach und die andere stark ist. Da eine weitere Einschränkung der Architektur auch mit Verwendung des Softwareframeworks genutzt werden kann, ein zusätzliches hinzufügen von weiteren Verknüpfungen aber nicht. Wird durch die Erkenntnis, dass einige ITSs sich auf die Benutzeroberfläche konzentrieren und andere auf die Datenbanken, sich für das offenere Modell entschieden.

Bei der weiteren Verfeinerung der Architektur wurde auf Probleme gestoßen, die sich auf den Einsatz von einzelnen Komponenten beziehen, welches dann automatisch bei Änderung von deren Aufgabenbereich auch andere funktionsnahe Komponenten betrifft. Die Anpassung des Lehrmaterials war anfangs nicht nur eine Komponente. Es gab zu der im Endresultat vorhandenen Anpassenkomponente eine Zweiteilung eben jener. Die Funktionalität wurde so aufgeteilt, dass eine Komponente für die Anpassung des Lehrmaterials zuständig war ohne Vorwissen über die Benutzeroberfläche oder wieviel Platz in dieser vorhanden ist. Die zweite Komponente würde das schon angepasste Lehrmaterial auf die Bedingungen der Benutzeroberfläche reduzieren und Entscheidungen über die Darstellung treffen, also das Material ausliefern.

Die Komponenten besaßen dabei zu sehr vernetzte Abhängigkeiten zu den anderen Komponenten. Die Grenze der Funktionalität der Komponenten war scheinbar falsch gewählt, wenn die Verantwortlichkeiten sich nicht so klar trennen lassen, dass die Abhängigkeiten reduziert werden können. Die erste Komponente generiert nur eine Vorauswahl, die viel zu groß ist und fängt schon an diese anzupassen. Obwohl nicht klar ist was davon verwendet werden kann. Die zweite Komponente kann das Lehrmaterial zwar korrekt anpassen, muss dafür aber zu viel über die Benutzeroberfläche wissen. Durch die Entwicklung der Szenen waren die Verantwortlichkeiten eindeutiger, denn nun war die Szene für die Anpassung des Lehrmaterials in sehr kleinen Maßstab verantwortlich und gleichzeitig war durch die Definitionen der Szenen der Maßstab und der Typ des Lehrmaterials eingeschränkt. So wurde die Funktionalität beider Komponenten reduziert und in einer Komponente, der Anpassenkomponente, zusammengeführt. Diese kann zwar weniger, das Ergebnis ist aber zielgerichteter. Nach Aufteilung des Lehrmaterials könnte auch Anpassenkomponente in ihrem neuen Funktionsanspruch wieder aufgeteilt werden um den jeweiligen Typ des Lehrmaterials genauer zu bedienen. Es gäbe dann eine informative und eine interaktive Anpassenkomponente. Wenn es aufgeteilt werden würde, dann müsste man auch andere funktionelle Komponenten aufteilen, wie die Bewertenkomponente. Inhaltlich kann man in der Definition immer feiner werden, doch es wurde erkannt, dass das Abstraktionsniveau der Anpassenkomponente in der Abstraktionsebene 4 keine weitere Aufteilung zulässt, weil sonst insgesamt zu viele Komponenten entstehen die zu weit weg von Abstraktionsebene 3 wären.

Es wurde über eine eigene ITS-Kernfunktionalität, für das Überprüfen der Eingaben für die Autorenfunktionen, nachgedacht. Es wurde sich dagegen entschieden, weil der Anspruch keine fehlerhaften Daten zu extrahieren eher in die Persistenzschicht passt. Bei offensichtlichen Syntaxfehlern kann dies die Datenbankschicht oder die Persistenzschicht schon abfangen, zum Beispiel durch einfaches Überprüfen eines Regex-Ausdrucks oder testen ob sich etwas im vorgegebenen Intervall befindet. Eine komplexere Überprüfung der Autoreneingabe die Zusammenhänge und teils die Semantik versteht könnte in die Funktionalschicht hineinreichen. Dann würden die Autoren nicht das letzte Wort haben, obwohl sie die Experten sind. Die meisten Flüchtigkeitsfehler sind mit der Persistenzschicht abgedeckt, einige semantische Flüchtigkeitsfehler könnten nicht erkannt werden, zum Beispiel wird ein „nicht“ vergessen. Probleme bei kurzen Eingaben mit kleinen Fehlern die semantisch eine große Auswirkung haben, sind die Annotationen. Die Kohärenz des Lehrmaterials wird auch nicht überprüft, zum Beispiel ob es einen ähnlichen Stil hat oder ob das passive und interaktive Lehrmaterial zusammen passen. Es wäre interessant den Effekt einer solchen Autorenüberprüfung genauer zu untersuchen.

Die Vorschlagenkomponente wird nur von wenigen Teilen des ITS verwendet. Die Überlegung war da, ob diese Komponente nicht gleich in die Profilgenerierung integriert werden sollte. Das Profil des Lernenden wird nicht in die Vorschlagenkomponente integriert. Es wird weitere Funktionalität geben, welche nicht zu den Hauptaufgaben eines ITS gehören, welche das Profil verwenden könnten. Ein Beispiel ist eine Funktionalität welche das Profil mit anderen vergleicht um ähnliche Lernende zu finden. Dies könnte dazu verwendet werden um passende Lernpartner oder passende Aufgaben zu finden, diese Aufgaben wären von ähnlichen Lernern schon als sehr gut in der Evaluation bewertet worden. Der Verantwortlichkeit der Funktionalität von der Vorschlagenkomponente und der Profilgenerierenkomponente sind einfach zu trennen und eine Zusammenlegung würde zu einer Aufblähung führen die sich bei einer Erweiterung einer der beiden Komponenten rächen würde.

Die Zusammenfassenkomponente scheint sich auf den ersten Blick in der Funktionalität sehr dem der Profilgenerierenkomponente zu ähneln. Deshalb ist die Frage ob die Funktionalität einer Zusammenfassung über den Lernenden direkt in die Profilgenerierenkom-

ponente integriert werden sollte. Das würde bedeuten die Zusammenfassung als Teil des Profils anzusehen. Ein Profil soll beinhalten, was der Lernende subjektiv mag und nicht mag, was der Lernende objektiv kann und nicht kann und wann der Lernende effizient ist. Eine Zusammenfassung hingegen listet objektiv auf was der Lernende abstrakt gemacht hat. Auch wenn die Profilgenerierung und die Zusammenfassung meist Hand in Hand gehen, so sind deren Anwendungsfälle verschieden. So erstellt ein Profil eine interpretierte Analyse des Lernenden. Eine Zusammenfassung versucht eine Interpretation in jedem Fall zu vermeiden. Nicht in jedem Fall in dem eine Zusammenfassung gebraucht wird, wird auch ein Profil verwendet. Die Zusammenfassung erlaubt es, wie bei Lernprogrammen üblich, sich selbst als Lernenden oder einem Experten eine Einschätzung zu bilden. Die Statistikszene ist dafür die Hauptszene und diese benötigt nur die Zusammenfassung. Es wurde sich für eine Trennung entschieden, weil es wichtige Fälle gibt, in denen nur die Zusammenfassung eine Rolle spielt. Da beide auch aus den vorhandenen Daten gewonnen werden können, lag die Umsetzung nahe, diese einfach nie zu speichern, sondern nur bei Bedarf ad hoc zu generieren. Einer der aufwändigsten Berechnungen eines ITS ist die Berechnung des Profils. Aus Gründen der Performanz, insbesondere wenn ein Vergleich der Eigenschaften des Lernenden über einen längeren Zeitraum getroffen werden soll bei dem für jeden Zeitpunkt ein Profil generiert werden muss, wurde sich für eine Speicherung entschieden. Dies erlaubt es auch ohne ITS die Zusammenfassungen und Profile einzusehen, welches einen entscheidenden Vorteil für die zentrale Forschungsdatenbank wäre. Zu guter Letzt musste dann die Entscheidung getroffen werden ob dann die Generierung der Zusammenfassung immer mit der des Profils einhergeht. Da eine Zusammenfassung nicht nur dann verwendet wird, wenn auch ein Profil gebraucht wird, wäre es wahrscheinlich nicht klug unnötig ein Profil zu generieren, wenn gar keines gebraucht wird. In der Praxis wird ein Profil aber in fast allen Fällen nötig sein und besitzt dann auch nicht mehr Daten als zum Zeitpunkt der Erstellung der Software. Wenn man also überlegt wann der Nutzer warten muss auf die Fertigstellung der Zusammenfassung, so könnte eine Generierung zum gleichen Zeitpunkt durch weniger Unterbrechungen sich durchaus besser in der Bedienung anfühlen. Weil nicht abgeschätzt werden kann, welches Design der Nutzerinteraktionen im Vergleich zum Speicherverbrauch von ungenutzten Profilen in Ausnahmefällen in denen kein Profil gebraucht wird besser ist, wurde die Trennung der Generierung beibehalten.

Die Programmfunktionalität trifft keine Unterscheidung der zeitlichen Verteilung der Kommunikation von einem Betreuer zu dem Lernenden. Damit ein Lehrer eingreifen kann, muss dieses gleichzeitig passieren. Beim Betreuer gibt es diese Bedingung nicht. Der Betreuer kann sich am selben Ort und/oder zum selben Zeitpunkt mit dem Lernenden auseinandersetzen. Ein Betreuer kann mit dem Lernenden über zeitversetzte Kanäle kommunizieren, wie zum Beispiel am häufigsten Vertreten durch ein Forum oder Textnachrichten. Eine zeitgleiche Kommunikation wie einem Audiotelefonat oder Videotelefonat hat andere Anforderungen an die Funktionalität. Die Anforderungen sind noch ähnlich genug als dass die Abstraktionsebene 4 nicht weiter aufgeteilt werden muss. Für den Fall, dass ein Betreuer am dem gleichen Ort ist, ist es irrelevant ob gleichzeitig kommuniziert wird. Weil der Betreuer ja schon am gleichen Ort ist, sollte die Kommunikation nicht künstlich verzögert werden.

Wenn der Lernende nicht weiter kommt stand die Option offen was gemacht wird. Eine Hilfefunktion hinzuzufügen, falls der Nutzer Probleme hat, war eine gute Idee. Doch ob diese Funktion eigenständig sein sollte oder nur ein Teil der Rückmeldung des Systems war eine schwierige Frage. Es gibt zwei Arten wie die Hilfe in den ITSs umgesetzt wurde: proaktiv und reaktiv. Bis zu diesem Zeitpunkt war die Architektur nur für ein reaktives System gedacht. Es wurde erkannt, dass mehrere Probleme dazu führen können, dass der Lernende proaktiv Hilfe braucht, darunter wurden die folgenden erkannt.

1. Der Lerner weiß nicht was er als nächstes tun soll

2. Der Lerner hat etwas nicht verstanden, ist sich dessen nicht bewusst
3. Der Lerner steckt in einer Richtung fest die nicht zum Ziel führt.

Das führte dazu, dass nicht nur die proaktive Hilfe benötigt wurde, sondern dass die Hilfe eine Funktionalität ist die unabhängig von der Rückmeldung vorhanden sein musste. Es gäbe die Option auch ein Log der der Interaktionen mit der Hilfe einzuführen. Es wurde sich aber dagegen entschieden, weil Hilfelog würde informatives und interaktives Logbuch in der Hilfe plötzlich vereinen. Deshalb ist die Interaktion mit der Hilfe während der informativen und interaktiven Lehrsitzungsszene in deren Logs integriert.

Die erste Zeit wurde sich mit der Benennung des Akteurs für die Person welche als Aufsichtsperson und beratende Hilfe den Lernern beisteht, wenn diese Probleme mit dem ITS haben, schwer getan. Denn es sollte verdeutlicht werden, dass diese Person keine Ahnung von dem Domänenwissen haben muss. Es sollten in dieser Arbeit möglichst viele Begriffe verwendet werden, welche in diesem Forschungsbereich üblich sind, das wäre in diesem Fall der Tutor. Doch dieser wurde nach weiterer Forschung verworfen, weil er zu sehr mit einer Lehrperson verknüpft wird. Schlussendlich wurde sich für einen einfachen Begriff entschieden: Betreuer als Übersetzung von Supervisor. Viele der Begriffe wirken offensichtlich, sind aber mit viel Bedacht getroffen worden. Einer der größten Hürden war die richtige Aufteilung und Benennung der Komponenten für die Analyse der Eingaben für die Rückmeldungen. Die Aufgabe einer Auswertenkomponente beinhaltet zwei Schritte. Verstehen was gemeint ist, wenn es zum Beispiel durch natürliche Sprache eingegeben wurde um mehr Freiheit zu gewähren, Bewerten ob die Antwort eine gute Antwort (in Bezug auf das was eigentlich erreicht werden soll) ist und eine Antwort darauf geben. Die Antwort geben wurde als Rückmeldenkomponente schnell ersichtlich. Die Aufteilung der anderen Bereiche war es anfangs nicht. Selbst nach der Aufteilung in eine Verstehen und eine Bewertenkomponente war der Name unklar. Eine Auswertung, eine Analyse oder eine Bewertung. Es kristallisierte sich mit der Entwicklung der benötigen Komponenten der Persistenzschicht heraus, dass es eine Bewertung ist, die möglichst objektiv eine Interpretation des Gemeinten liefert. Eine Auswertung beruht weniger auf einer Interpretation und eine Analyse erfordert mehr Querverweise zwischen den vorhandenen Daten. Bewertungen sind der erste Schritt der Interpretation der Eingaben des Lernenden und damit der Analyse des Profils und besseren Anpassung des Lehrmaterials und Verhalten des ITS. Deshalb gehört die reine Eingabe und was gemeint war bei Freitexten ist zu dem „Interaktiven Logbuch“ und die Sortierung der Antwort in Klassen zu Bewertungen. Die Aktionen des Interaktiven Logs lassen sich nur im Kontext bewerten. Ob eine Antwort richtig oder falsch war und inwieweit sie gut oder schlecht war sind entscheidende Fragen. Dies trifft auch auf die Interaktion während der Präsentation des Informativen Lehrmaterials zu, nur nicht ganz so stark. Das Informative Logbuch zählt zum Beispiel auf, was wie lange und in welcher Reihenfolge gesehen wurde. Man kann allerdings auch bewerten ob dieser Schritt gut oder schlecht war. Eine Interaktion mit dem informativen Lehrmaterial ist zum Beispiel welches Lehrmaterial sich der Lernende angeschaut hat. Ein Lehrmaterial zu betrachten, was man schon gesehen hat, kann als schlecht bewertet werden. Das ergibt nur dann Sinn, wenn die Präsentation nicht nur in eine Richtung gehen kann die vorher festgelegt ist. Ein einfaches Zurück würde schon ausreichen. Dann ist die Funktion zur Bewertung aber auch sehr simpel, aber dafür gibt es auch Variationen, zum Beispiel ob es schlechter ist wenn man mehr als eine „Folie“ zurück geht. Für die Evaluationen tritt der Kontext auch ein. Denn eine zu schwere Lehrsitzung, mit einem Inhalt der eigentlich als zu leicht eingestuft wurde, wird anders bewertet als umgekehrt.

Die Verstehenkomponente selbst besitzt keine Komponente in der Persistenzschicht die die Interpretation der natürlichen Sprache separat einordnet. Die Interpretation der Eingabe des Lernenden könnte in einem eigenen Log gespeichert werden. Da in einem System nur eine Interpretation vorhanden ist, wäre es übertrieben dies weiter zu spezifizieren. Falls mehrere Interpretationen zu einer Eingabe üblich werden ist die Interaktive Log

Komponente nicht falsch bezeichnet oder verknüpft, sie besteht dann nur aus mehr noch kleineren Komponenten.

Über welche Komponente in der Funktionalschicht die Speicherung der Eingaben von dem Lernenden in „Informatives Logbuch“ und „Interaktives Logbuch stattfindet war eine späte Schwierigkeit in der Softwarearchitektur. Sollte die Helfenkomponente diese Funktionalität bekommen? Die Helfenkomponente sollte nur dann zum Einsatz kommen, wenn eine Hilfefunktion benötigt wird. Ohne Helfenfunktion müsste man trotzdem immer die Helfenkomponente aufrufen, wenn der Lernende eine Eingabe getätigt hat. Dies ist selbst bei einer passiven Helfenfunktion der Fall, diese sollte aber nur dann benötigt werden, wenn der Nutzer es aktiv wünscht. Die Helfenkomponente ist also dafür nicht geeignet. Die Prozesssteuerungskomponente sollte unabhängig von der Persistenzschicht sein. Die Programmeinstellungen sollte unabhängig von den Lehrfunktionen sein. Die Bewertenkomponente wird immer dann aufgerufen wenn der Lernende etwas eingegeben hat in der „Interaktiven Lehrsitzungsszene“, dies würde also gut passen. Doch dann sollte dies auch für die „Informative Lehrsitzungsszene“ gelten. Ist eine Auswertung für angepasstes informatives Lehrmaterial immer geeignet? Da Bewertungen für informatives Lehrmaterial geschaffen wurden weil die geringste Definition der Bewertung eine Anerkennung das etwas geleistet worden ist dazu passt, ist dies auch eine gute Stelle die Funktionalität zum speichern zu gewährleisten. So stellt sich diese Frage auch für die Speicherung der Evaluationen. Dafür die Profilgenerierenkomponente zu nehmen, weil diese schon Zugriff auf die Evaluationen hat, ist verwirrend. Wenn man wegen des Namens denkt warum die Evaluation eine Profilschätzung braucht, obwohl es nur die Evaluationen speichern soll, würde es das Vertrauen in eine gute Benennung, Definition und Schlussendlich in die Umsetzung reduzieren. Es bleiben noch die Bewertenkomponente oder die Programmfunktionskomponente. Die Programmfunktionskomponente würde damit dann Zugriff auf alle Komponenten mit roher Eingabe haben, also Informatives Log, Interaktives Log und Evaluationen. Dies würde aber die Programmfunktionskomponente unnötig viel Zugriff erlauben, der eigentlich nicht in ihr Ressort passt. Da Bewertungen für informatives Lehrmaterial und Evaluationen einen Mehrwert bieten, ist dies auch eine gute Stelle die Funktionalität zu speichern zu integrieren.

Die richtige Definition für Verknüpfungen der Komponenten kostete sehr viel Zeit. Die vier problematischsten Verknüpfungen werden folgend aufgeführt. Die erste Verknüpfung die schwierig war, war die Vorschlagenkomponente zu der Hilfeszene. Die Hilfeszene muss Inhalte, die passend zur der aktuellen Lehrsitzung sind anzeigen, dass kann heißen das Lehrmaterial angezeigt wird, welches als Grundlage für das Lehrmaterial in der Lehrsitzung dient. Welches Lehrmaterial dann angezeigt wird, könnte die Vorschlagenkomponente für die Helfenkomponente herausfinden. Mit dieser Idee gibt es Probleme. Die Vorschlagenkomponente generiert Vorschläge direkt mit Hilfe des Profils, die Helfenkomponente würde aber mit der Idee nicht das Profil sondern nicht vollumfänglichen Zugriff auf das benötigte Wissen haben und damit nicht selbstständig Material finden das relevant für die Hilfe ist. Zudem sind die Vorschläge, die die Vorschlagenkomponente macht, nicht an die aktuelle Lehrsitzung gerichtet, sondern als eigene Lehrsitzung gedacht. Auch deshalb braucht man eine Helfenkomponente. Die Helfenkomponente kann diese Funktionalität nicht auslagern. Das ist in so weit nicht so schlimm, als dass die Algorithmen zur Suche von passenden Lehrmaterial nicht ohne Anhaltspunkte anfangen, sondern dass durch die schon bestehende Lehrsitzung und das gerade aktuelle Lehrmaterial sehr viel konkrete Daten vorliegen, zu was Hilfe benötigt wird. Wird die Idee nicht umgesetzt dann ist auch eine Begründung für die Verknüpfung schwierig. Also wurde die Hilfeszene nicht mit der Vorschlagenkomponente verknüpft. Das zweite Problem war, ob die Autoritätenkomponente auch auf die Bewertungen zugreifen kann. Autoritäten sollen den Lernenden überwachen und dafür ist eine Einschätzung womöglich eine gute Idee. Doch die Autoritäten sollen gar nicht in die Bewertungen des ITS einsehen. Der Lehrer bildet sich sein ei-

genes Bild vom Verständnis des Lernenden und der Betreuer ist nicht für die Einschätzung des Lehrwissens genügend ausgebildet. Deshalb wurde sich gegen eine Verknüpfung der beiden Komponenten entschieden. Die dritte Frage ist die Verknüpfungen zwischen der Kollaborierenkomponente zu den Bewertungen. Die Kollaborierenkomponente braucht keine Verknüpfungen zu den Bewertungen, weil die Bewertungen nur dann benötigt werden, wenn diese über das Netzwerk synchronisiert werden müssen. Es sollen so wenig Daten wie möglich über das Netzwerk an Daten geschickt werden sollen. Ein anderes ITS kann die Bewertungen eigenständig nachbilden, dies ist nicht sehr effizient für die Rechenleistung aber dafür für die Datenrate. Wenn die ITSs nicht exakt gleich sind in der Generierung der Bewertungen, dann erzeugt dass einen Unterschied in den Bewertungen. Dies würde aber auch dafür sorgen, dass keine Daten übertragen werden die nicht zum System passen, sondern passende Daten immer neu erzeugt werden. Es wurde entschieden, dass für die Schwachstelle die Bewertungen über das Netzwerk zu schicken unnötig kompliziert wäre. Die Schwachstelle wäre es, mehrere ITSs innerhalb einer Lehrsitzung für Lernende die Kollaborieren zu verwenden. Es wurde sich daher dafür entschieden, dass die Kollaborierenkomponente keinen Zugriff auf die Bewertungen hat.

Die letzte Frage stellte sich über die Helfenkomponente und die Informationen die sie über den Nutzer haben dürfte. Eine einfache Helfenfunktion kann nur auf die vorliegenden Informationen von der Lehrsitzung zugreifen. Die Anpassungsfähigkeit und bessere relevant der Antworten an den Nutzer kann durch mehr Daten erreicht werden. Da die Helfenfunktion als einer der Wege angesehen wird, auf den Lernenden auch proaktiv zuzugehen und sich an sein Verhalten, nicht nur an seine Fähigkeiten anzupassen, wurde der Helfenkomponente mehr Verknüpfungen zugeteilt.

Die Folge der Komponenten und wie diese in speziellen Fällen aufeinander in ihrer Funktionalität zu einem Ziel aufbauen, kann unterschiedlich gestaltet werden. Es stellt sich die Frage ob die Diagramme interaktiv sind. Für die eigentliche Doktorarbeit ist dies nicht von Vorteil, da diese nur mit statischen Bildern auskommen muss. In der App zur Erstellung eines vorgefertigten ITS mag es nützlich gewesen sein, da es die Architektur besser zu erklären mag. Da von der geplanten ITS-Erstellungssoftware abgesehen wurde, war der Aufwand mit nicht genügend Nutzen versehen und es wurde sich dagegen entschieden. Eine Markierung aller Verknüpfungen insbesondere in der Abstraktionsebene 4 wäre, wenn man nur in der visuellen Ansicht ist, sehr hilfreich.

Es sollte keine Kommunikation zwischen den Szenen direkt stattfinden, denn es sollte nur eine Szene gleichzeitig aktiv sein. Eine Kommunikation zwischen allen Szenen würde zum Kontrollverlust der Kommunikation führen und Kopplung verursachen. Auch ist es so, dass bei den anderen Schichten die unteren Komponenten sich untereinander nicht kennen. Es ist also besser die Entscheidung zu welcher Szene eine andere folgt nicht von der Szene selbst getroffen wird. Man könnte es der nächsthöheren Komponente überlassen, mit dem Problem dass man nicht ein System mit nur Komponenten der untersten Schicht entwickeln kann. Die Szenen sollen sich nicht untereinander kennen. Weil man dann nicht jede mit jedem verbinden würde. Wenn man aber nur die verbindet, die auch voneinander wissen sollten, dann ist damit implizit auch der Prozess in die Architektur integriert. Um diesen zu trennen gibt es die Prozesssteuerungskomponente. Diese verbindet abhängig davon, was sie als Prozess definiert hat bestimmte Szenen miteinander. Es wird das Coordinator-Pattern mit der Prozesssteuerungskomponente implementiert. Da die Prozesskomponente sowieso schon den nächsten Schritt und damit häufig auch implizit die Entscheidung der im ITS-Prozess relevanten Szenen übernimmt, wurden ihr Aufgabenbereich um die Wahl der folgenden nächsten Szene ergänzt. Wenn man nicht nur die Komponenten der weniger abstrahierten Schichten nimmt, sondern auch höhere, dann können diese Daten auch im ITS, in der Temporärschicht oder in der jeweiligen Szenenkomponente gespeichert werden (Lehrszenen oder Zusatzszenen).

V.2.d. Entscheidungen im Softwareframework

Die Schwierigkeiten und komplizierten Entscheidungen die getroffen werden mussten für ein neues Softwareframework, wurden durch die große Vorarbeit in der Softwarearchitektur stark reduziert. Somit treten für das Softwareframework viel weniger diskussionswürdige Themen in den Vordergrund.

Die Begründung warum die Wahl auf die Programmiersprache Swift gefallen ist, wurde im Kapitel IV.1.a. ausführlich behandelt. Zusammenfassend ermöglicht es die Programmiersprache neuere Konstrukte in der OOP umzusetzen, welche mit den Vorstellungen von UML der Architektur sehr gut einhergehen. Eine Transformation von UML in konkreten Quellcode war auch zum Dank der Fähigkeiten der Programmiersprache recht problemlos. Dies trifft natürlich auch auch andere Programmiersprachen, vornehmlich neuere, zu. Es war die Idee, dass eine zukunftsorientiertes Softwareframework auch eine zukunftsorientierte Programmiersprache verwendet. Unter den aufsteigenden beliebten neuen Programmiersprachen stach Swift mehr als heraus. Die Unterstützung sowohl von der Firmenseite als auch von der Community die Fragen beantworten und Blogs zu Problemen schreiben war wesentlich größer als andere aufsteigende Programmiersprachen wie Rust oder Dart.

Einer der größeren Schwierigkeiten war, ob eine variable oder feste Struktur und Anzahl an Elementen angeboten wird. Es wurde mit einem variablen Ansatz der Softwarearchitektur begonnen, bei dem es nur wenige vorgeschriebene nötige Komponenten gibt, die eingebunden werden mussten. Der Rest der Komponenten wäre unnötig für diesen Anwendungsfall und sollte damit auch gar nicht erst in der Softwarearchitektur erscheinen. Anstatt für jede Kombination ein eigenes Softwareframework zu erzeugen, war die Idee, ein Programm zum Erstellen von schon vorgefertigten ITSs zu entwickeln in der das Framework zusammengestellt wurde. Eine FRAMEWORK-Datei wird wohl immer nötig sein, da sie einfach einzubinden ist und keine Möglichkeiten bietet, sie fehlerhaft zu verändern. Wenn diese sich als flexibel genug darstellt, dann reicht es als einzige Möglichkeit. Die Entscheidungen wurde erst später in der Entwicklung geändert, deshalb wird im Kapitel V.2.b. auf diese Entscheidungen als Irrweg tiefer eingegangen. Diese Entscheidung wurde mit einer zweiten Variante des Softwareframeworks leicht wieder entgegengewirkt. Mit der detaillierten Definition der Architektur kam es bei der Entwicklung der Softwarearchitektur zu einem Problem. Einerseits sollen die detailliertesten Komponenten in dem Framework vertreten sein, aber die Komplexität der Verschachtelung der Komponenten mit höheren Abstraktionsgrad außen vor zu lassen. Gemeinsame Funktionalitäten würden zwischen den einzelnen Komponenten die zu einer Komponente höherer Abstraktionsebene nicht mehr in dieser zusammengefasst werden können. Zum Beispiel könnte die Domänenwissenkomponente nicht mehr gemeinsame Funktionalität für alle in ihr enthaltenen Komponenten abbilden, weil in der Variante mit nur den detailliertesten Komponenten nicht mehr vorhanden wäre. Es ist möglich nur eine Abstraktionsebene zu implementieren, zum Beispiel nur die detaillierten Komponenten. Die Frage war ob diese Option gut genug wäre um sie auch umzusetzen. Die wird aber nicht empfohlen, weil in den höheren Abstraktionsebenen die Funktionalität die geteilt wird hochgehoben wird. Es ist wahrscheinlich, dass Komponenten einer geringeren Abstraktionsebene, welche in einer Komponente vereint sind auch gemeinsame Funktionalität besitzen werden. Da bei kleineren Projekten alle Abstraktionsebenen zu implementieren als abschreckend angesehen werden kann und jede Abstraktionsebene auch für sich genommen funktioniert, wurde nachträglich eine zweite Variante erstellt. Welches kaum die Nachteile des variablen Aufbaus von mehreren individuell zusammengestellen Softwareframeworks hat, aber dafür einen großen Vorteil davon: Unnötige Komplexität verringern. Wenn man also wenig gemeinsame Funktionalität hat und eine Verdopplung von Verwaltungsaufgaben unwahrscheinlich ist, ist die zweite Variante effektiver.

Die Wahl zwischen einer dynamischen und einer statischen Bibliothek als Verlinkung des Softwareframeworks musste getroffen werden. Eine dynamische Bibliothek benötigt nur eine Verlinkung zu einer schon vorhandenen Datei. Statische Bibliotheken liefern die FRAMEWORK-Datei einfach mit dem Programm mit. Die zweite Variante ist einfacher, sicherer und bietet kaum Nachteile. Es wurde die statische Variante genommen. Bei bekannten Computerumgebungen könnte die Wahl für ein dynamisches Framework, welches vom Betriebssystem geliefert wird, überwiegen. So könnte man sicher gehen, dass alle ITSs auf dem Computer wirklich die exakt gleiche Versionsnummer des Frameworks verwenden. Die Wahrscheinlichkeit dass mehrere Versionen des Frameworks existieren und dann mehrere ITSs auf einem Rechner dieses verwenden, ist in absehbarer Zeit verschwindend gering. Auch ist die Größe des Frameworks, wenn sie mit der Applikation als statische Bibliothek mitgeliefert werden, nicht ausschlaggebend, weil jedes hochauflösende digitale Foto größer ist.

Die zugrundeliegende Softwarearchitektur des Softwareframeworks basiert auf dem Komponentenmodell, dies erlaubt es flexibel die einzelnen Komponenten auszutauschen. Allerdings ist die Rahmenstruktur der Komponenten, sowie die Komponenten selbst mit Beschreibung ihrer Funktionalität, als auch die Abhängigkeiten zueinander schon festgelegt. Damit wird zwar die Handlungsfreiheit in der Entwicklung des Softwareframeworks stark eingeschränkt, aber es wird ihm viel an Arbeit abgenommen, wie das System aufgebaut werden soll. Es stellte sich die Frage wieviel zusätzliche Arbeit das Softwareframework übernehmen sollte.

Jede Komponente könnte ein oder mehrere komplette Implementierungen haben, die es dem Entwickler auch noch abnehmen die Funktionalität zu programmieren. Es wurden sich die Unterschiede der Komponenten für die vorhandenen ITSs angeschaut und ob die Implementierung einer Komponente sich stark voneinander unterscheiden. Das Problem ist es natürlich auch, die zugrundeliegenden Unterschiede der genutzten Softwarearchitekturen und der Plattformen und Werkzeuge. Es konnte für die Implementierung der jeweiligen Komponenten keine einheitliche kleine Anzahl gefunden werden, die fast alle Anforderungen abdecken würden. Dies würde in den Forschungsbereich 2 fallen, weil eine Komponente gebaut werden sollte, diese aber an die Bedingungen nicht für einen Prototypen sondern für viele ITSs angepasst werden müsste. Dazu hätte eine Analyse der Implementierung und Softwareanforderungen in den jeweiligen Einrichtungen beitragen müssen und das hätte aus dem Fokus dieser Arbeit geführt. Eine Evaluation und Anpassung an den Bereich würde sogar in den Forschungsbereich 3 reichen. Die Freiheiten sind durch die Softwarearchitektur schon eingeschränkter und es sollte davon abgebracht werden, eine komplette Standardimplementation zu nehmen, auch wenn sie ungünstig ist, nur weil sie da ist. Die Qualität hätte für eine wirklichen Einsatz sehr robust und von hoher Qualität sein müssen. Produktionsqualität einer kompletten Komponente wäre höchstens für einen Spezialfall zeitlich Umsetzbar gewesen und dass wäre dann eigentlich nur die benötigten Funktionen vom Prototypen in das Framework zu verschieben. Es wurde sich deshalb entschieden nur eine rudimentäre Implementierung anzubieten, die zwar eigenständig funktioniert, deren Funktionalität aber minimal ist.

Eine weiterführende Idee ist auch die Implementierung von Komponenten aus den äußeren Schichten. Dies würde das Problem der Einschränkung der Einsatzfähigkeit sogar noch schlimmer machen, da man sich dann auch gleich auf Datenformate und wie die Daten darin intern aufgebaut sind festlegt. Vorteilhaft könnte es für die zentrale Forschungsdatensammlung sein, die Struktur der Datenerhaltungsschicht festzulegen. Dann wäre es möglich mit nur einem Format alle Fälle abzudecken und damit die Einheitliche Implementierung zu maximieren. Leider ist das illusorisch. Es gibt nicht ohne Grund so viele Plattformen. Viele erfüllen ihre eigene Nische. Selbst wenn zwei Plattformen der gleichen Nische nur eine Implementierung bekommen, so ist häufig so, dass nicht beide den gleichen Standard unterstützen. Jede gewählte Technologie macht die Anzahl einsetzba-

rer Plattformen kleiner. Ob mit JSON, XML, SQL oder anderen Formaten gespeichert wird oder sogar wie die Tabellen, Attribute und Werte aussehen dürfen zu bestimmen, legt konkret Dinge fest, die für einige Anwendungsfälle und ITSs sehr ungünstig sind. Um solch ein Problem anzugehen müsste man also eine Vielzahl an Optionen bieten, die an die Bedürfnisse der wichtigsten Anwendungsfälle, Domänen und Plattformen angepasst sind. Oder man entwickelt einfach einen Prototypen. Die Wahl ist auf den Prototypen gefallen.

Technisch lässt es sich mit Strukturen oder Klassen die Anforderungen der Protokolle für die Komponenten umsetzen. Viele Verknüpfungen sind zwischen den Komponenten der unterschiedlichen Schichten vorhanden und diese Verknüpfungen werden mit Referenzen implementiert. Die Klasse ist selbst ein Referenztyp und wird bei Zuweisung eine Referenz verwenden. Es ist damit simpler wenn man einfach ein Objekt einer Klasse erzeugt und darin gestaffelt nach den Abhängigkeiten diese eingeben kann ohne extra Syntax zur referenzierung oder Wrapperobjekte wie bei Strukturen. Dies fällt besonders dann ins Gewicht wenn man die Transformation aus dem Kapitel IV.1.b. betrachtet, in dem noch schwache Referenzen verwendet werden um auch die Vernetzung von Komponenten unterschiedlicher Abstraktionsebenen implementiert. Deshalb wurde sich entschieden die Protokolle Klassen als einzige Option für die Implementierung derer zu erlauben. Damit kann immer von einer Referenzierung ausgegangen werden und nicht von einem Kopiervorgang bei Zuweisung oder Parameterübergabe. Die Abbildung der Architektur kann komplett nur durch Protokolle bewerkstelligt werden. Ganz ab von einer Implementierung der Klassen zu lassen würde dazu führen, dass entweder leere Referenzen (nil oder null) übergeben werden müssten und diese Funktionalität komplett wegfällt oder dass rudimentäre Implementierungen die nur das gröbste erfüllen verwendet werden. So wurde also eine Implementierung geliefert die nur minimale Funktionalität hat und damit nur dann ausreicht, wenn man keine eigenen Anforderungen hat. Einer der größten Klassen an Fehlern, den „Null pointer exceptions“ und den damit verbundenen Code für dessen Sicherheitsmaßnahmen wurde komplett umgangen.

Mit den Referenzen kam ein weiteres Problem in Bezug auf das Softwareframework, welches nur die konkretesten Komponentenprotokolle implementiert: Wie sie verknüpft werden. Es wurde sich für die simpelste Variante entschieden, dass eine zusätzliche Datei diese Arbeit übernimmt. So bleiben die einzelnen Komponenten fast exakt gleich im Code und sind nicht belastet mit Wissen und Verknüpfungen von Komponenten. Zudem werden alle Verknüpfungen in einem Schritt und in einer Datei gesammelt. Allerdings führt das zu einer Klasse, die nicht in der Architektur erwähnt wurde. Eine Eigenheit der Variante die vertretbar ist.

V.2.e. Diskussion über den Prototypen

Der Grund warum eine native Applikation als Implementation des Prototypen gewählt wurde ist simpel. Wenn es fehlerlos gemacht ist, merkt der Lernende nur wenig vom Unterschied zwischen einer nativen Applikation oder zum Beispiel einer Web-App. Eine Webapplikation hat aber Beschränkungen, die für Probleme sorgen können und es können mehr Fehler bei einer Web-App auftreten. Die meisten Fehlerquellen, die nichts mit der Softwarearchitektur oder dem Softwareframework zu tun haben, hat die Web-App. Auftretende Fehler in einem zweigeteilten über Netzwerk agierenden System sind vielfältig und haben auch nichts mit dem Softwareframework direkt zu tun. Probleme mit der Internetverbindung, die unterschiedlichen Browser die unterstützt werden müssen, ein Aufspaltung in Client-Server und so weiter, werden durch eine native lokale macOS Applikation gänzlich vermieden. So konnte sich auf das für diesen Prototypen wichtige Strukturierung konzentriert werden ohne den Rahmen zu sprengen: Umsetzung der Architekturanforderungen an ITSs mittels des Frameworks. Der Prototyp selbst braucht dabei den

vollen Zugriff auf die Tastatur und damit auf tiefere Schichten den Betriebssystems, welches ein Client in einem Browser einfach nicht hat. Zudem agiert der Browser selbst als eine Zwischenschicht für die Eingaben und verändert diese teilweise. Bei einer vorverarbeitete Eingabe kann man sich nicht sicher sein aus welcher Quelle sie stammen und wie genau sie sind. So könnten sie von einem Skript sein. Eine native Applikation kann direkt die gesendeten Keycodes der Tastatur von dem Betriebssystem abfangen, bevor sie weiterverarbeitet werden.

Es wurde sich für den Prototypen auch für die Programmiersprache Swift entschieden, weil das Softwareframework schon in Swift geschrieben wurde. Für ein Beispielimplementierung mittels des Softwareframeworks ist es am verständlichsten, wenn sie einheitlich die gleiche Sprache verwenden. Andere Optionen wären Objective-C, C++ oder einer anderen Programmiersprache welche mit dem Framework dann über C APIs als kleinsten Nenner kommunizieren würde und damit eine unnötige Komplexität erzeugt. Der Prototyp selbst ist keine Entwicklung für den die Programmierung einen gleichwertigen Einfluss auf andere ITSs erwartet wird, auch wenn er Open Source gestellt wird. Es war wichtiger die Einheitlichkeit zu gewährleisten als einen großes Anwenderspektrum.

Es sollte für den Prototypen eine Domäne gewählt werden, die für ein ITS gut geeignet ist. Diskrete Eingaben lassen sich semantisch leichter zuordnen, führen aber häufig zu einer zu starken Einschränkung der Antwortmöglichkeiten. Ein Beispiel sind die Tests für Humanmediziner deren sehr großen Handlungsmöglichkeiten in der realen Welt auf genau fünf Antworten reduziert werden, bei dem die Antworten bereits ersichtlich sind. Also sollte der Handlungsspielraum in der realen Welt in der Domäne mit dem des im interaktiven Lehrmaterial abgefragten kongruent sein und dazu in einer Größendimension, welche für einen Prototypen in einer Doktorarbeit passend ist. Zudem sollte kein Domänenexperte integriert werden, dessen Domäne weit ab von dem Thema der Doktorarbeit ist, also sollte es etwas sein, mit dem sich die Informatik beschäftigt. Trotzdem wäre es schön, wenn die Zielgruppe groß ist. Tastaturen sind Werkzeuge, die nicht nur jeder Informatiker täglich verwenden muss, sondern haben auch eine geringe Hardwarekomplexität und deren Verhalten basieren auf Treibern. Deren Domäne ist klein und lässt sich leicht abgrenzen. Der Aufgabenbereich ist einfach abgesteckt und die Interaktionen mit der Tastatur abzählbar und sehr häufig. Die generierten Daten über den Nutzer sind vielzählig, auch bei geringer Nutzungsdauer. Anpassungen des interaktiven Lehrmaterials sind relativ einfach zu implementieren. Schwierig hingegen ist es herauszufinden was in welche Richtung angepasst werden muss und welche Schlüsse gezogen werden müssen um die richtigen Tipps zu geben. Die zentralen Fragen einer ITS-Implementierung bleiben bestehen und die Analyse des Lernendenwissens eine Hürde. Insgesamt eine perfekte Domäne für diese Arbeit.

Die Benutzerschnittstelle ist ein Bereich der ITS-Entwicklung in denen Human-Computer-Interface (HCI)-Designer die beste Wahl treffen sollten. Es wurde mit besten Wissen und Gewissen eine simple Benutzeroberfläche erschaffen. Damit diese aber auch zukünftig verändert und gezeigt werden kann ohne den Prototypen zu starten gibt es einen grafischen Editor der deklarativ die Gestaltung in XML-Code speichert. Es hat den Vorteil, dass sich keine Logik in die vom Konzept her reine Ansicht einschleichen kann. Deshalb wurde sich für STORYBOARDS als Dateiformat für die Benutzeroberflächen entschieden.

SQL ist der Standard wenn es um Datenbanken geht. SQLite ist schnell und kann Unmengen an Daten effizient speichern. Diese Daten beinhalten das Lernendenwissen. Der Lernende darf aufgrund der Eventualität von Schummeln nicht auf diese Daten zugreifen und Autoren müssen auf diese Daten nicht zugreifen, deshalb sind sie auf Performanz und nicht auf Lesbarkeit oder Nutzerfreundlichkeit getrimmt. Auch wenn die Tabellen, SQLite-Typen und Benennungen der Spalten verständlich gehalten wurde, so kann man davon ausgehen, dass zum Beispiel Forscher sich für die Daten ein SQLite-Verwaltungsprogramm installieren und sich etwas mit SQLite beschäftigen, um an die Daten zu gelangen und zu exportieren. NoSQL ist für diese Art nicht so effizient, sondern eher doku-

mentenbasiert. SQLite als eine Form von SQL ist simpler und effizienter als zum Beispiel MySQL oder Postgres mit dem Nachteil, dass es weniger Funktionen gibt. Alle Funktionen die benötigt wurden waren aber schon in SQLite, deshalb konnten auf die anderen Funktionen verzichtet werden. Eine Textdatei ist der Einfachste Zugang für Autoren selbst wenn kein Autorenwerkzeug genutzt wird. Da kein extra Autorensystem gebaut wurde, sondern es in den Prototypen integriert wurde und das Lehrmaterial simpel verändert werden sollte, wurde dieser Teil der Daten einfach in eine UTF-8 TXT-Datei geschrieben. So ist es plattformunabhängig möglich, dass der Autor das informative Lehrmaterial anpasst. Wenn eine veränderte Datei wieder im „Content"-Ordner innerhalb des „App"-Ordners verändert wird, aktualisiert sich auch der Lehrinhalt des Prototypen.

Obwohl es Lehrthemen gibt, wurde zusätzlich das Lehrmaterial in Einzeleingabe und Serieneingabe von Tastenkombinationen, die gedrückt werden sollen, unterschieden. Es wurde deshalb so gemacht, weil die Serieneingabe dem realen Anwendungszweck in den meisten Fällen gleich kommt. Insbesondere bei Wörtern, kann das Auge schon Buchstaben lesen, die noch nicht getippt wurden. Diese Form des interaktiven Lehrmaterials war aber zu ungenau, wenn der Lernende noch eine geringe Fertigkeit mit bestimmten Tasten auswies. Deshalb wird die Einzelanzeige und -eingabe für den Anfänger und die Serienanzeige für Fortgeschrittene angewendet. Dazu gibt verschiedene Stufen der Unterstützung von Anzeige des korrekten Fingers bis zum Hervorheben der richten Tasten und wo sie sich auf der Tastatur befinden. Diese Stufen wurden dazu eingeführt um wie bei Aufteilung in Einzel- und Serieneingabe der eine feinere Granularität der Unterstützung zu liefern, welche bei einem höheren Lernendenwissen hinderlich wird.

Anfangs wurde gespeichert, wann welche Modifikatoren gedrückt wurden. Der Datensatz, der pro Tastendruck in den Log gespeichert wird, für die Performanz bei hoher Tippgeschwindigkeit, geringem Energieverbrauch und geringer Datengröße soll minimal bleiben. Deshalb wurden daraus berechenbare Daten oder Daten, die keinen Beitrag zur Analyse des Lernenden führten, aussortiert. Das Ziel ist es, mit den Daten in einem Profil dem Lernenden Tipps zu geben. Die Tipps mit den Modifikatoren würden sich auf Probleme mit dem Timing der Modifikatortasten in Bezug zur Nichtmodifikatoren in Tastenkombinationen fokussieren. Da es aber vier Modifikatoren gibt mit sieben Modifikatortasten und durch die Art der Speicherung mit nur Bools und Strings in SQLite für jeden Modifikator ein eigenes Feld erzeugt werden musste, war der Druck hoch nützlich zu sein. Die entstandenen Tipps die daraus gewonnen wurden waren leider nicht so nützlich. dass die Zeit gespeichert wird wann der Modifier aktiviert wurde ist zu sagen man sollte ihn nicht so weit im Voraus drücken und dass ist wohl nicht mal ein guter Tipp. Es gab ein weiteres Problem bei Modifikatoren: Standardmäßig haben die doppelt angebrachten Modifikatoren auf der rechten und linken Seite der Tastatur mit einem ANSI- oder ISO-Hardwarelayout jeweils für eine Modifikatorfunktion den gleichen Keycode. So hat die rechte und linke Umschalttaste den gleichen Keycode. Es benötigt weiteren Programmcode der etwas komplizierter ist um dies herauszufinden. Mit zusätzlichen Timings wurde dies weiter strapaziert. Deshalb wurde sich nachträglich wieder dagegen entschieden den Zeitstempel für das drücken einer Modifikatortaste mit zu speichern.

V.2.f. Weitere Ziele der Resultate im Forschungsbereich 1

Um diese Arbeit als Vorlage für einen Standard verwenden zu können, wären noch viele formale Schritte und Treffen notwendig. Ein Ziel wäre es diese Richtung weiter zu verfolgen und sich die Arbeit zu machen, die notwendige Überzeugungsarbeit des Konzepts zu leisten. Eine große Institution für Standarisierung (wie zum Beispiel IEEE) könnte einen verfeinerten Standard ITS-Forschung gezielt vorantreiben. Das vorgestellte Konzept ist in der vierten Abstraktionsebene komplexer als die üblichen Standards, dennoch kann das Konzept in vielen Bereichen genauer spezifiziert werden. Dann sollte man aufpassen,

dass nicht zu stark konkretisiert wird, als dass damit die Allgemeingültigkeit verloren gehen könnte. Ein Ansatz wäre es für die verschiedenen Domänen eine konkretere Version des Konzepts zu erstellen und gleichzeitig das allgemeine Modell einzuhalten.

In dieser Arbeit wurde absichtlich nicht der Fokus auf die äußeren Schichten gelegt, eine genauere Definition der Interaktionsschicht. Designrichtlinien für die Benutzeroberflächen der verschiedenen Szenen würden bei dem Erstellen von Ansichten helfen. Regeln sollten beachtet werden, wenn man die Oberfläche für Autoren, Lernende, Betreuer und Lehrer entwirft. Welche Farben, Anordnungen und Größen sind für die jeweiligen Rollen relevant. Was motiviert und was ermüdet. Das sind Fragen die sich bestimmt auch international unterscheiden. Viele Regeln lassen sich von Designkonzepten ableiten und sollten auf ihre Wirksamkeit auch im Bereich der ITS getestet werden.

Auf der anderen Seite sind Regeln für den Aufbau der Datenbanken genauso wichtig. Auch hier lassen sich aus der Datenbankforschung bestimmt viele Grundregeln anwenden. Doch die genaue Beschreibung der Forschungsdaten und deren Verwaltung der Daten von unterschiedlichen ITSs ist sehr viel spezieller. Ein ITS ist eine perfekte Möglichkeit gleichartige Tests an Lernenden durchzuführen, weil sie exakt wiederholbares Verhalten der Lehrer liefern. Doch es nützt nur dann etwas wenn die Daten vielzählig sind. Eine Forschungsdatenbank ist einfach definiert, doch wie schafft man es, dass die Daten freiwillig im Sinne der Forschung veröffentlicht werden ohne den Datenschutz zu verletzen. Der Aufwand für die Transformation von Datenquellen aus unterschiedlichen ITSs sollte so gering wie nur nötig sein. Auch hier wäre ein Standard sehr hilfreich um zu definieren um Vorteile für die gesamte Lehrforschung zu erzeugen.

Bisher wurde nur die Struktur in einem Softwareframework konkretisiert, in dem sich eine Prozesssteuerungskomponente befindet. Abhängig von der Programmiersprache könnte es programmatisch umgesetzt werden, dass der Lehrprozess überprüft und eingehalten wird, ohne eine Softwarearchitektur zu verwenden. Also auch den Prozess unabhängig als einfach einzubindende Vorlage zu implementieren. Dazu kann man ausführliche Definition aller möglichen Prozesse definiert werden. Jeder Prozess hat seine Vor- und Nachteile und für bestimmte fein definierte Prozesse gibt es womöglich Anwendungsfälle für die sie sich am besten eignen. So kann zum Beispiel auch in Verbindung damit die Wiederauflebung der Idee einer Tabelle mit den Nutzungsfeldern in der Konfiguration der Komponenten der Architektur und einem speziellen Prozessauswahl Goldstandards für ein Aufgabenfeld liefern. Bestimmte Parameter beeinflussen dabei die Wahl welcher feiner definierte Prozess und Konfiguration des Konzepts am besten wären.

Weitere Lehrforschung die ITSs implementieren und alle mit der gleichen Softwarearchitektur oder sogar Softwareframework erzeugt wurden und damit das gleiche Modell beinhalten würden auch Rückschlüsse auf Verbesserungen geben, die zurück in den Forschungsbereich 1 fließen würden. Durch eine Erleichterung der Implementierung kann eine Einrichtung, welche nicht so viele Programmierer zur Verfügung hat, Zeit und Geld sparen.

V.2.g. Anwendung im Forschungsbereich 2

Dies ist eine Forschungsarbeit, welche die Vorarbeit für einen Standard definiert der im Forschungsbereich 1 angesiedelt ist. Das bedeutet, dass eine Verwendung der Architektur in eingesetzter Lehrsoftware das primäre Ziel ist. Das Ergebnis dieser Doktorarbeit gibt es in unterschiedlichen Detail- beziehungsweise Abstraktionsebenen. Für jemanden der einfach nur ein ITS implementieren will stellt sich die Frage wie sinnvoll es ist, überhaupt eine Architektur zu verwenden und warum es dann so kompliziert sein muss. Es läuft auf eine Time-Cost-Analysis (TCA) hinaus. Bei dem es durchaus vorkommt, dass ein Programm absichtlich nicht mit einem Anspruch auf die korrekte Anwendung von Softwaretechniken entwickelt wird und trotzdem besser geeignet ist. Dies gilt vor allem für Pro-

gramme die ein großes Risiko darstellen und deren Komplexität nicht so hoch ist. Die Investition an Entwicklungszeit wird unter Umständen keine Einnahmen bringen. Zu Marktforschungszwecken kann es eine Methode sein um herauszufinden, welches Produkt sich zu welchem Preis verkaufen lässt. Ohne Einhaltung wichtiger Prinzipien der Softwaretechnik wie SOLID [2002 Martin], Tests schreiben oder einhalten einer Architektur kann die Entwicklung eines Prototypen vielfach beschleunigt werden. Viele Vorteile gehen dadurch aber auch verloren. Darunter zählt die Wartbarkeit, die Zuverlässigkeit, die Sicherheit, die Skalierbarkeit und die relativ einfache Weiterentwicklung der Software. Alles Vorteile die sich langfristig bewähren. Ein gutes ITS ist in seiner Entwicklung anspruchsvoller als eine Prototyp-Applikation zu Marktforschungszwecken, sodass sich eine Entwicklung langfristig ohne Softwaretechnik wahrscheinlich nicht rentieren wird. Desto komplexer eine Software allein durch deren Aufgabenstellung gestaltet werden muss, umso schwieriger wird es ohne Prinzipien der Softwaretechnik dieses Ziel überhaupt zu erreichen und auch längerfristig aufrecht zu erhalten.

Eine Lücke zwischen dem ersten und dem zweiten Forschungsbereich wurde mit einem Softwareframework gefüllt. Dieses wurde erst nach vollständiger Vollendung des Konzepts entwickelt. Somit wurde ausgeschlossen, dass plattformabhängige Konzepte der eigenen Softwareframeworkimplementation die abstrakten Modelle beeinflusst. Es kann sein, dass deshalb noch kein so komplexe allgemeingültige Architektur entworfen wurde, weil es für eine einzelne ITS-Implementation wenig Vorteile bringt. Sich die Arbeit zu machen die perfekte Abstraktionstiefe zu finden, um nützlich aber gleichzeitig nicht zu konkret zu sein und die Modellierung von Modellen die in ITSs im Allgemeinen angewendet werden, kann nur schwer in die Entwicklung eines konkreten ITS gesteckt werden. Man könnte für ein einzelnes ITS auch eine sehr konkrete Softwarearchitektur entwerfen mit integrierten Softwareartefakten. Ein konkretes fertiges ITS kann in der Lehre direkt eingesetzt werden, während ein Pre-Standard nur dann für die Entwickler eines ITS wirkliche Vorteile bringt, wenn es jemand anderes einsetzt. Da das ein Risiko darstellt sind ad-hoc entwickelte ITS am häufigsten vertreten. Die Zwischenvariante der Entwicklung eines Softwareframeworks gibt es immerhin wenige Male. Dies macht allerdings auch erst dann Sinn wenn man mehr als ein ITS damit entwickelt und ist dann leider in der Modellierung auf eine Plattform beschränkt. Besonders weil das Ergebnis neue Wege geht kann dies abschrecken. Wie groß die Wahrscheinlichkeit ist dass es eingesetzt wird kann man nicht sagen. Es hängt von weit mehr Faktoren ab als der Qualität der Arbeit. Dies ist noch nicht das Ende der Entwicklung des ersten Forschungsbereiches von ITS in den Prozessen und Softwarearchitekturen. Für ein breiteres Anwendungsspektrum ist für andere Anwendungsbereiche für andere Plattformen eine Implementierung der Softwarearchitektur noch in anderen Programmiersprachen vorteilhaft. Andere Umsetzungen des Softwareframeworks wären auch mit der gleichen Programmiersprache für eine gewisse Plattform begrüßenswert. Die Anwendung der Konzepte für bestimmte Entwicklertypen kann damit verbessert werden. Die Entwicklergemeinschaft ist aktuell durch die Benutzung unterschiedlicher Technologien in native Entwickler und Webentwickler gespalten. Auch wenn es andere Entwicklertypen gibt, zum Beispiel die Systementwickler, so fällt die Entwicklung eines ITS in die Arbeit der beiden Erstgenannten. Ein Softwareframework mit Fokus für Webentwickler hätte andere Prioritäten um leicht eingebunden und genutzt werden zu können.

Der Prototyp selbst ist eine Implementation der Konzepte und befindet sich damit im Forschungsbereich 2. Um als eine einsatzfähige Software in der Lehre, zumindest für Forschungszwecke eingesetzt zu werden, wäre es geeignet ein von Experten und Didaktikern ausgearbeitet Konzept zu verwenden. Dann kann eine Lehrstrategie mit wissenschaftlicher Grundlage gewählt werden. Dies könnte die Aufarbeitung des Domänenwissen mit Integration von mehr Wissenstypen als Text erfordern. Wissenselemente aus Bildern, Videos oder dergleichen würden die Abwechslung Motivation und Anpassungsfähigkeit an

die Geschmäcker der Lernenden verbessern. Eine Erweiterung des interaktiven Lehrmaterials um kleine Geschichten oder visuelle Spiele würde die Motivation bestimmt auch positiv beeinflussen ohne einfach nur ein Spiel zu machen. Zur Vergrößerung der Zielgruppe zu erhöhen ist eine Implementierung in mehreren Sprachen ein schneller Weg.

Der dritte Forschungsbereich erlaubt noch viel Forschungsarbeit, denn noch sind wir weit davon entfernt ITS als reguläres Werkzeug in der Lehre einzusetzen. Das gilt vor allem für Fächer, die sich nicht so einfach in konkrete Regeln einbetten lassen. Es wurde Kontakt zu den verschiedensten Anwendungsbereichen geknüpft um herauszufinden, welche Bedürfnisse an ein ITS gestellt werden. Insbesondere wurde dies in den Bereichen durchgeführt, welche versuchen ITSs zu verwenden, wie zum Beispiel der Finanzsektor, der Medizinsektor, die Lehre oder das Militär. ITSs haben ihren Einsatzzweck und es muss den Nutzern der Unterschied zu Expertensystemen und Trainingssystemen klar gemacht werden. In einigen Situationen wo heute Expertensysteme oder Trainingssysteme eingesetzt werden, wären ITSs besser geeignet. Da dieser Forschungsbereich so wenig vertreten ist und diese Arbeit sich im Forschungsbereich 1 ansiedelt, wird nur bis zum Forschungsbereich 2 diskutiert.

V.2.h. Ethische Aspekte und die Konsequenzen der Arbeit

Das Ziel ist bei der Lehre mit einem ITS nicht die Lehrer zu ersetzen, sondern in Situationen bei dem ein Lehrer keine Aufmerksamkeit hat oder kein Lehrer vorhanden ist, der Lernende nicht alleine Lernen muss. Das Lernen mit einer Software ist hierbei die Lösung mit einem ITS. Eine Software erhöht die Inklusion und Integration. Zum Beispiel für Menschen mit einem anderen kulturellen Hintergrund, bei dem die sozialen Aspekte weniger relevant sind. Das kann auch Menschen helfen, welche Probleme damit haben sich in eine Gruppe einzufügen, nicht in der Gruppe lernen wollen oder gar eine soziale Phobie haben.

Bei einer vermehrten Nutzung mit einer Software und nicht mit Menschen, könnte das Lernen zu sehr auf das Fachwissen beschränkt sein und das Lernen von Wissen vernachlässigen, welches nicht explizit im Lehrmaterial widergespiegelt wird. Soft-Skills und Persönlichkeitsentwicklungen, die dem Lernenden helfen könnten sich außerhalb des Fachwissens sich zurecht zu finden.

Interaktionen mit realen Gegenständen können nicht einfach in einem ITS eingebunden werden, weil dieses kein inhärentes Verständnis seiner Umwelt hat. Die Daten und die Interaktionen müssen für ein digitales System digital vorliegen. Damit ist man auf Interaktionen gebunden, welche der Software bekannt sind, wenn diese Interaktionen für den Lernvorgang relevant sind und von dem virtuellen Lehrer auch bemerkt werden sollten.

Datenschutz ist ein weiterer wichtiger Punkt und ein ständiger Kritikpunkt bei der Entwicklung von künstlicher Intelligenz von heute. Da die Adaptivität von einem ITS auch von einem genauen Lernendenwissen herrührt und dazu dessen Datenbank mit vielen Datenpunkten des Lernenden gefüllt wird, werden immer einige Daten über den Lernenden gesammelt die unter Umständen den Lernenden selbst identifizieren könnten oder sensibel sind. Desto mehr Datenpunkte gesammelt werden, also desto länger und intensiver der Lernende mit dem ITS lernt, umso mehr weiß das ITS über den Lernenden. Wenn jetzt zur weiteren Forschung ausgewählte oder alle Daten an einen zentralen Server anonym gesendet werden sollen, so lassen große Datenmengen bei bestimmten Personen doch immer gewisse Rückschlüsse zu. Im Wirklichkeit wird es dann eine Pseudonymisierung. Ohne ein Wissen über den Lernenden ist es praktisch unmöglich die gesteckten Ziele für ein ITS und das Konzept zu erreichen. Um es trotzdem zu verbessern, kann Datenschutz dadurch verbessert werden, indem man die Daten auf dem Endbenutzergerät verschlüsselt und diese auch nur dort entschlüsselt werden können, dies schließt dann leider auch viele der Szenarien aus, bei denen die Akteure nicht am gleichen Ort sind. Eine andere

Option ist es bestimmten Personen und Geräten auch die Entschlüsselung zu erlauben, für Forschungszwecke oder durch die entfernte Benutzung. Dies ist die übliche Vorgehensweise bei Nutzerdaten, die auf zentralen Servern bei Firmen lagern. In diesem Fall muss man auf die Datensicherheit der zentral verwalteten Server vertrauen und Daten nur mit Ende-zu-Ende-Verschlüsselung übertragen um Man-in-the-middle-Attacken auszuschließen. Eine dritte Option ist die teilweise Randomisierung von einem kleinen Prozentsatz der Daten, die die Ergebnisse eines großen Datensatz nicht zu sehr verzerren. So kann man sich nie wirklich sicher sein ob ein spezieller Datensatz auch wirklich so aufgenommen wurde oder zufällig verändert wurde, während die Gesamtaussage gleich bleibt. Letzten Endes zeigen viele Datendiebstahl-Ereignisse der Vergangenheit dass jede Datensicherheit umgangen werden kann und Datenschutz nur dann zu 100% gewährleistet ist, wenn man die Daten erst gar nicht speichert. Der Sinn ist es also gar nicht eine 100% Datensicherheit zu gewährleisten, sondern nur benötigte Daten in pseudonymisierter Form zu speichern und dabei es den Eindringlingen es schwer genug zu machen, dass es sich nicht lohnt. Der Vorteil von den Daten die vom ITS gesammelt werden ist, dass sie selten von brisanter Natur sind. Sie umfassen weder ethnische, finanzielle oder private Daten des Nutzers und sind daher per se schon ein uninteressanteres Ziel für Hacker als vergleichsweise andere Datenbestände.

V.3. Ausblick

Man kann das Projekt als Grundlage für die vergleichbare Forschung der ITSs und KI verwenden. Die Bereiche welche verglichen werden könnten, wenn ein System verwendet werden würde, wären Dinge wie Laufzeit, Benchmarks oder die Funktionsweise der ITSs.

Es benötigt noch eine Menge an Entwicklungsarbeit damit ITSs ihrer Aufgabe gewachsen sind. Besonders die Entwicklung muss von neuen angepassten ITSs schneller, weniger komplex in der Entwicklung/Wartung und kostengünstiger sein, um flächendeckend eingesetzt zu werden. Dies kann mit der Weiterentwicklung anderer Bereiche der Forschung einen großen Sprung machen, wie zum Beispiel der aktuelle Trend zum Machine Learning unter anderem das Verstehen und Artikulieren der natürlichen Sprache Vorteile im Dialogsystem von ITSs werden kann. Verschiede große Technologiekonzerne veröffentlichen für ihre Platform Softwareframeworks - Apples „MLKit“, Googles „Tensorflow“, Microsofts „Azure ML“, Amazons „Amazon Machine Learning“ - welche genau diese Lücke schließen könnten, aber bisher noch viel Arbeit und das sammeln von Beispielen benötigen. Dabei wird häufig Machine Learning (ML) verwendet. Bei ML ist es schwierig Erklärungen für die getroffenen Entscheidung zu finden, es ist in etwa so als ob automatische Entscheidungen in einer Black-Box getroffen werden. Die meisten Algorithmen von ML basieren nicht auf der Nachvollziehbarkeit von Symbolorientierten Algorithmen wie man es bei den ITS-Typen vorfindet, sondern es sind sehr viele Zahlen (Gewichte) die durch Vorkommen bestimmter Muster beim lernen an einer Trainingsmenge mit bekannten Ergebnis angepasst werden. ML verstärkt häufig den Bias durch Selbstverstärkung. Bei einem automatisierten Lehrer wäre hier nicht durch das trainierte Modell eindeutig klar auf was dieser seinen Fokus setzt. Unter Umständen auf Zusammenhänge die nicht in eine Bewertung einfließen dürften, wie zum Beispiel die Bewertung abhängig von der Tageszeit machen, weil die meisten guten Antworten nicht nachts um drei Uhr kommen. Ohne Nachvollziehbarkeit können immer Fehler im Verhalten enthalten sein, bei wichtigen Entscheidungen ist es sehr ungünstig für den Lernenden. Bereiche die wenig formal definiert sind wie die Freitexteingaben, Zeichnen oder Sprach- und Videoaufnahmen werden schon heute von ML erfolgreich analysiert und können sinnvolle Einsatzzwecke für ITSs bieten. Einige Bereiche, wie das Kunstverständnis, sind aber noch in weiter Ferne. Die zukünftige Revolution der KI wird Software effektiver machen bei geringeren Produktionskosten [2017 MAKRIDAKIS].

Die Fähigkeit der ITSs adaptiv zu sein ist noch nicht hoch genug, insbesondere wenn eine Analyse der nicht vorgefertigten Antworten des Lernenden betrachtet wird. Zeitgleiche Anpassung des Verhaltens des ITS während der Interaktion mit dem Lernenden sind sehr selten. Dies sollte noch weiter geführt werden, anstatt nur das Wissen des Lernenden zu modellieren. Es sollten auch viele weitere Daten über den Lernenden gesammelt werden, um damit dem ITS eine bessere Datenlage für die Einschätzung zu geben. Desto mehr Daten gesammelt werden, umso besser muss auch die Datensicherheit werden. Letztendlich führt die Auswertung von mehr Daten zu einem ITS, dass zusammenhängende Probleme besser erkennen kann und der Lernende Dinge beschreiben kann ohne sie explizit zu benennen, weil es implizit klar ist. Ein einfaches Beispiel für eine Lernendeaussage wäre dann: „*Es* ist wohl anders als *letztes Mal*“. Ein Satz der auf eine versteckte Frage hindeutet, bei dem weder „es“ noch „letztes Mal“ explizit sind. Erst bei solch einer Stufe des Verständnisses schaffen wir einen automatisierten Lehrer, der eigenständig ein Dialog vorantreibt um dem Lernenden proaktiv dann Hilfe zu bieten, wenn er sie braucht und nicht vorher.

Die Interaktionsmöglichkeiten von ITSs sind bisher auf die Standardschnittstellen begrenzt: Tastatur, Maus, Bildschirm und unter Umständen auch Audio. Es wäre erstrebenswert diese Schnittstellen zu erweitern um Mikrofone, Kameras, Virtual Reality, Aug-

mented Reality oder Barrierefreiheit. Eine häufig vernachlässigte Funktion ist die Kollaboration von mehreren Lernenden mit einem ITS bei dem dem System bekannt ist, dass es mehrere Nutzer sind und jeden von ihnen und ihre Interaktion von eben jenem System einzeln analysiert werden kann. Die kurzfristig wirkungsvolle Implementation wäre: Jegliche lernrelevante Kommunikation der Lernenden untereinander und mit dem System über ihr jeweilige Benutzerschnittstelle zu machen. Zukunftsmusik wäre diese Kommunikation direkt abzugreifen um die persönliche Kommunikation nicht zu unterbinden, sei es durch Kameras/Mikrofonen oder anderen Mitteln. In diesem Sinne könnte man auch, wie von einigen ITSs bisher schon grob integriert den emotionalen Zustand erkennen [2008 NEJI, AMMAR and ALIMI]. Nicht zu vergessen, dass in der Hardwareentwicklung besonders die Größe immer weiter schrumpft, bei dem ein ITS nicht auf einem Computer läuft, sondern auf einem Smartphone.

Um noch einen Ausblick zu schaffen welche andere Architektur in Zukunft für ITS noch wichtig werden könnte auf das schon kurz in der Arbeit eingegangen wurde: Agentensysteme. Eine zukünftige Umsetzung könnte so aussehen, dass die Komponenten als Agenten implementiert werden: Eine Gruppe von Agenten würde jeweils eine Lerneinheit selbständig anhand der Ressourcen vorbereiten, welche für die Domäne für ihn verfügbar ist, eine andere Gruppe von Agenten analysiert wie es um das Wissen des Lernenden steht, eine weitere Agentengruppe kommuniziert mit dem Lernenden, eine weitere Agentengruppe nimmt die Auswertung der Lösungen von dem Lernenden vor und so weiter. So könnten gleichzeitig mehrere Agenten versuchen mit mehreren Methoden das beste Ergebnis zu erstellen und würden dem Lernenden diese zur Auswahl gegeben. Der Vorteil wäre dann in der Konkurrenz verschiedener implementierter Agenten, welche gleichzeitig zur Laufzeit geeignete Hilfe und Material anbieten. Die beste Lehrmethode würde sich mit der Zeit automatisch durchsetzen und eine Evolution der Lehre begünstigen.

Die Bewertung eines ITS beruht bisher auf einen Experten, der nicht nur die Lehre und die Domäne kennen muss, sondern auch ITSs im Allgemeinen. Diese Bewertung ist nicht standarisiert. Eine Bewertung ist durch die Flexibilität und Vielfältigkeit bei ITSs an sich schon schwer und unterscheidet sich abhängig von dem Bewertendem noch zusätzlich. Ein ITS-Standard würde eine Bewertung von ITSs vereinfachen, auch weil es wenige Experten gibt. Noch besser wäre hingegen ein Bewertungsstandard für ITSs einzuführen.

Wünschenswert wäre dass die Softwarearchitektur in Zukunft in mehrere Softwareframeworks implementiert wird. Jedes Softwareframework mit seinen eigenen Schwerpunkt mit ihren eigenen Vor- und Nachteilen, angepasst an die benötigten Anforderungen. Es wäre sinnvoll die in dieser Doktorarbeit vorgestellte Softwarearchitektur weiterzuentwickeln und sie als Anstoß für ein neu entwickeltes Standard-Grundmodell einzusetzen. Eine andere Möglichkeit ist sie genauer zu spezifizieren ohne sie dabei für ungültig zu erklären. So kann auch eine genauere Spezifikation für zum Beispiel eine bestimmte Domänen durchaus sinnvoll sein. Eine Weiterentwicklung wäre auch den Lehrprozess feiner zu beschreiben, zum Beispiel für eine bestimmte Domäne oder mit Annahmen im Aufbau des ITS. Zudem ließe sich auch der Prozess selbst an einen Lernenden anpassen, wenn sich herausstellt, dass ein Lehrprozess sich bei bestimmten Situationen für die Lerneffizienz ändern sollte. Dies könnte vor allem bei einem sehr feingranularen Prozess auftreten. Bei einer Adaptivität des Prozesses würde die Prozesssteuerungskomponente auch auf die Profile zugreifen. Es wäre auch möglich außerhalb dieser Arbeit ein tiefgreifendes Modell für die beiden äußeren Schichten zu erstellen. Darunter fällt die Benutzeroberfläche und die Datenbanken. Diese wurden absichtlich außen vor gelassen, weil sie nicht der Fokus der Arbeit sind und weitreichendes Wissen und Analysen von ITS nicht aus einer Perspektive der Softwarearchitektur erfordern, sondern aus Sicht eines Datenbankingenieurs oder Human-Computer-Interface-Designers. Es sind beides große Forschungsrichtungen in der Informatik, welche den Rahmen der Arbeit gesprengt hätten. Es wäre möglich ein oder mehrere Designs der ITS-Benutzeroberflächen als grundlegende Konzepte zu ent-

werfen und über Studien herauszufinden, welche Benutzeroberflächen für welche Domäne effizient ist. Dies müsste sowohl für Autoren als auch für Lernende ausgewertet werden, zudem sollten auch vorhandene ITS nach ihrer Benutzeroberfläche untersucht werden. Auch auf der anderen Seite der vorgestellten Schichtenarchitektur, den Datenbanken, würden Auswertungen über die vorhandene Nutzung von Datenbankentechnologien sinnvoll. Es müsste herausgefunden werden, welche Ansätze in den Strukturen von Datenbanken von ITS umsetzbar und performant sind, wenig Speicherplatz verbrauchen und zu dem Gedankenmodell von Autoren passt.

Es ist sehr schwer sich ein potenziell effektiveres System für die Lehre vorzustellen. Ein ITS beinhaltet schon ein semantisch verknüpftes Netz des Lehrinhalts, eine Vorstellung davon was der Lernende weiß und was er nicht versteht und eine Funktionsweise die es dem System erlaubt sich an den Lernenden und die Umgebungsgegebenheiten anzupassen. Zusätzliche Eigenheiten sind Dinge die ein menschlicher Lehrer nicht kann wie die Synchronisation der ITSs untereinander. Diese und andere Herausforderungen für ITS erlauben noch eine langjährige Forschung an diesem Thema, die hoffentlich nicht aneinander vorbei, sondern zielgerichtet hin zu einem Standard die Ergebnisse bündeln und damit alle ITSs im Gesamten verbessert. ITSs fallen in die gleiche Sparte der Nützlichkeit als Lehrer, wie digitale Assistenten (Google Assistant / Siri / Cortana / Bixby) als Sekretär. Man hätte lieber einen Echten, aber sie sind an eine Person angepasst und kosten wenig genug, um als Werkzeug nützlich zu sein. Auf das ITSs in Zukunft die Fähigkeiten besitzen von denen einige Menschen schon zur Schulzeit träumten.

VI. ANHÄNGE

VI.1. Verzeichnisse

VI.1.a. Abkürzungsverzeichnis

Abkürzung	Bedeutung
ABI	Application Binary Interface
AEH	Adaptive Educational Hypermedia
AH	Adaptive Hypermedia
AIED	Artificial Intelligence in Education
CAI	Computer Assisted Instruction
CBI	Computer Based Instruction
CBT	Constraint Based Tutor
CMS	Content Management System
CSCL	Computer Supported Collaborative Learning
DSL	Domain Specific Language
E-Learning	Electronic Learning
GBL	Game Based Learning
HCI	Human Computer Interface
IDE	Integrated Developement Evironment
ILLS	Intelligente Lehr-/Lernsysteme
IoC	Inversion of control
ITS	Intelligent Tutoring System
JaBIT	Java Based Intelligent Tutoring
JVM	Java Virtual Machine
LAMS	Learning Activity Management System
LMS	Learning Management System
ML	Machine Learning
MTT	Model Tracing Tutor
OOP	Objektorientierten Programmierung
PC	Heimcomputer
TEL	Technology Enhanced Learning
TPM	Tutoring Process Model
UI	User Interface
UML	Unified Modeling Language
UX	Benutzerinteraktion
VLE	Virtual Learning Environment
Wiki	Wissensmanagementsoftware

VI.1.b. Abbildungsverzeichnis

Abbildungskürzel	Inhalt
R-II4c-1	Grobe Einteilung des ITS-Prozesses [2017b GRAF VON MALOTKY, NICOLAY and MARTENS]
R-II4c-2	Domänenwissen initialisieren
R-II4c-3	Pädagogikwissen initialisieren
R-II4c-4	Lernendenwissen initialisieren
R-II4c-5	Lehreinheit generieren
R-II4c-6	Informatives Lehrmaterial unterrichten
R-II4c-7	Interaktives Lehrmaterial unterrichten
R-II4c-8	Evaluation des Lernenden bekommen
R-III1b-1	Beispiel der Peer-To-Peer-Architektur
R-III1b-2	Beispiel der Client-Server-Architektur
R-III1b-3	Beispiel der Schichtenarchitektur
R-III1b-4	Beispiel der Model-View-Controller-Architektur
R-III1b-5	Beispiel der Multitier-Architektur
R-III1b-6	Beispiel der Component-Based-Architektur
R-III1b-7	Beispiel der Domain-Driven-Design-Architektur
R-III1b-8	Beispiel der Message-Bus-Architektur
R-III1b-9	Beispiel der Repository-Architektur
R-III1b-10	Beispiel der Pipes-And-Filters-Architektur
R-III1b-11	Beispiel der Publish-Subscribe-Architektur
R-III1c-1	ITS-Softwarearchitekturen pro Jahr die gefunden wurden
R-III1c-2	Prozentuale Anteil der Anzahl der klassischen Komponenten in den gefundenen ITS-Softwarearchitekturen
R-III1c-3	Prozentuale Implementierung der klassischen Komponenten in den gefundenen Softwarearchitekturen
R-III1c-4	Anzahl der implementationsspezifischen Komponenten in den gefundenen ITS-Softwarearchitekturen
R-III1c-5	Neue Komponenten in den gefundenen Architekturen
R-III1c-6	Berücksichtigung eines Autors für das Domänenwissen oder Pädagogische Wissen („knowledge engineer“) in den gefundenen ITS-Softwarearchitekturen
R-III1c-7	Anzahl unterschiedlicher Namen für die klassischen Komponenten in den gefundenen ITS-Softwarearchitekturen
R-III1d-2	Architektur welche viel zu implementierungsspezifisch ist [2003 CROWLEY and MEDVEDEVA]
R-III1d-3	Nichtformelle Visualisierung einer ITS-Architektur [2006 LEE, CHO and CHOI]
R-III1d-4	Schlechte Verbindungen zwischen den Komponenten [2006 KOJIMA and MIWA]
R-III1d-5	Abstraktion von Architekturen mit schlechten Verknüpfungen
R-III1d-6	Schicht 3 der LTSA, IEEE 1484.1 [2003 IEEE]
R-III1d-7	Keine echte Architektur [1998 SELF]
R-III1d-8	Anwendung einer klassischen Architektur [2013 AHUJA and SILLE]
R-III1e-1	Hat eine Prozesssteuerungskomponente in den gefundenen ITS-Softwarearchitekturen
R-III1e-2	Komponenten der klassische ITS-Softwarearchitektur in einem Komponentenmodell und zusätzlicher Prozesssteuerungskomponente
R-III1f-1	3-Schicht-Architektur als Model-View-Presenter
R-III2b-1	Kombination der überarbeiteten klassischen Architektur mit MVP
R-III2b-2	Wichtige Schicht ist die mittlere mit der fehlenden Funktionalität
R-III2c-1	Abstraktionsebene 1 der Softwarearchitektur
R-III2c-2	Schichten der Softwarearchitektur (Abstraktionsebene 2)
R-III2c-3	Die obersten zwei Abstraktionsebenen
R-III2d-1	Klassische Komponenten in dem neuen Schichtmodell
R-III2d-2	Komponenten der zweiten und dritten Abstraktionsebene
R-III2d-3	Obersten drei Abstraktionsebenen
R-III2e-1	Szenen in der Temporärschicht mit Abhängigkeiten
R-III2e-2	Beispielaufbau der Ansichten in einem ITS
R-III2f-1	Komponenten der Funktionalschicht in der dritten und vierten Abstraktionsebene
R-III2g-1	Komponenten der Persistenzschicht in dritter und vierter Abstraktionsebene
R-III2h-1	Architekturkomponenten und deren Abhängigkeiten innerhalb der gleichen Komponente
R-III2h-2	Softwarearchitektur aller Komponenten bis zur vierten Abstraktionsebene
R-IV1c-1	Beispiel einer Verschachtelung aus zwei Schichten
R-IV1c-2	Referenzen zur Umsetzung von Verschachtelung in Swift

Abbildungskürzel	Inhalt
R-IV1d-1	Ordnerstruktur des Projekts mit Variante 1
R-IV1d-2	Ordnerstruktur des Projekts mit Variante 2
R-IV1e-1	Screenshot mit Markierung der wichtigen Stellen in wichtiger Reihenfolge
R-IV2b-1	Komponenten des Prototypen
R-IV2e-1	Abbildung des Hauptmenüs (unterstützt helles und dunkles Thema)
R-IV2e-2	Beispiel des informativen Lehrmaterials
R-IV2e-3	Beispiel des interaktives Lehrmaterials
R-IV2e-4	Beispiel der Evaluation
R-IV2e-5	Beispiel der Statistik
R-V2b-1	Bildschirmfoto der ITS-Erstellungssoftware

VI.1.c. Tabellenverzeichnis

VI.2. Tabelle der klassischen Komponentennamen

Zitierung	Name vom Domänenwissen	Name vom Lernermodell	Name vom Pädagogikmodul	Name vom Benutzeroberfläche
[1984 CLANCEY]	Domain Knowledge Base		Teaching Knowledge; Interpreter	
[1984 O'SHEA, BORNAT and DU BOULAY]	Teaching generator	Student model; Student history	Teaching strategy	
[1985 ANDERSON, BOYLE and REISER]	Domain expert		Tutoring knowledge; Bug catalogue	User interface
[1990 NWANA]	Expert knowledge module	Student model module	Tutoring module	User interface module
[1991 ASHLEY and ALEVEN]	Domain Expert	Student Model	Pedagogical Modul	
[1991 BURNS, PARLETT and REDFIELD]	DOMAIN EXPERT	STUDENT MODEL	INSTRUCTIONAL EXPERT	INTELLIGENT INTERFACE
[1992 LUSTI]	Expertenmodul	Studentenmodul	Unterrichtsmodul	Kommunikationsmodul
[1992 PUPPE]	Wissensmodell	Tutandenmodell	Didaktik-Komponente	Benutzerschnittstelle
[1994 IKEDA and MIZOGUCHI]	Expertenmodul	Studentenmodell	Tutormodul	Benutzerschnittstelle
[1995 VASANDANI and GOVINDARAJ]	Expert Module	Student Module	Instructional Module	Tutorial Interface; Simulator Interface
[1996 DE BARROS COSTA and PERKUSISCH]	Human experts society			Interface Agent
[1997 CORBETT, KOEDINGER and ANDERSON]	Domain Knowledge	Student Model	Pedagogical Module	Problem solving Environment
[1997 SONG, HAHN and TAK]			GOES; ExBug; Curriculum Network	
[1998 SELF]	Domain Model	Student Model	Tutoring Model	
[1998 SIEMER and ANGELIDES]	Domain Model	Student model	Teaching Model	User interface
[1999 ALPERT, SINGLEY and FAIRWEATHER]	Expert Solver	Student Model	Tutorial Module	UI Proxy / Socket Communicator
[1999 LELOUCHE]	Domain EXPERT MODULE	MODEL MANAGEMENT MODULE	TUTORIAL MODULE	USER INTERFACE AND DIALOGE
[1999 SHAW, GANESHAN and JOHNSON]		Store for case, student, persona, references and simulation parameters	Reasoning engine	Simulation engine and GUI; Animated persona
[2000 CAPUANO, MARSELLA and SALERNO]	Knowledge Library	ABITS Data Base		Web Browser
[2000 FREEDMAN, PENSTEIN and RINGENBERG]			Tutorial Planner / GUI Interpreter (Andes)	User Interface Manager
[2000 GERTNER and VAN LEHN]	Solution Graph; Physics Problem Solver; Physics Rules; Problem Definition	Student Model	Help System	Workbench
[2000 HARRER]	Expertenmodul	Studentenmodul	Tutormodul	Kommunikationsmodul
[2000 LOS ARCOS, MULLER and FUENTE]		Session log	Assistant; Tutor	STUDENT & INSTRUCTOR INTERFACE
[2000 MATSUURA, OGATA and YANO]	Contents & Templates	Personal Knowledge; Action Log		UI
[2000 MAYO and MITROVIC]	CBM; constraints	student models	Pedagogical module; databases, problems, solutions	Interface

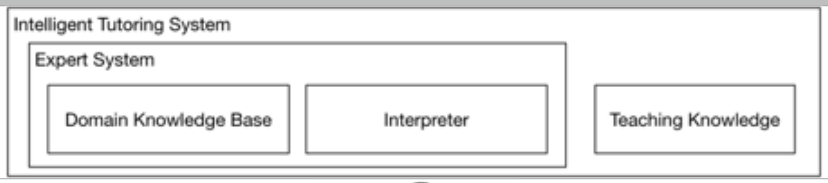
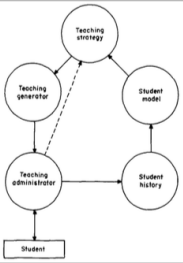
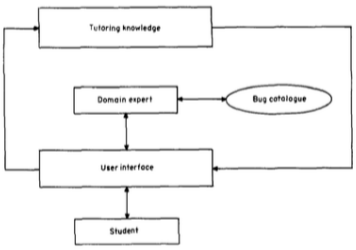
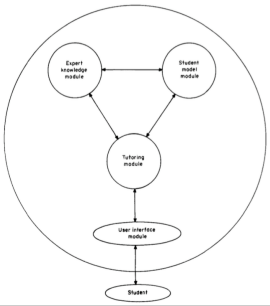
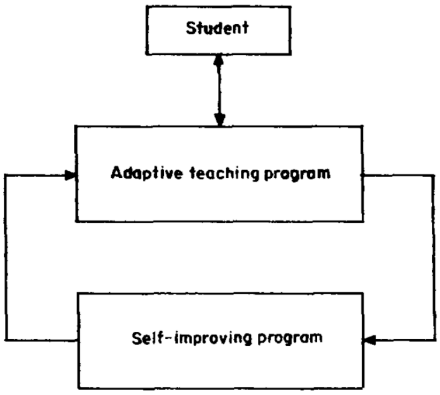
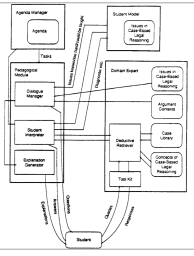
Zitierung	Name vom Domänenwissen	Name vom Lernermodell	Name vom Pädagogikmodul	Name vom Benutzeroberfläche
[2000 MAYO, MITROVIC and MCKENZIE]	Problem Database	Student Model	Pedagogical Module; Constraint Database; Student Modeller	User Interface
[2000 MICHAUD, MCCOY and PENNINGTON]	Domain KB	User Model	Error Identification Module; Response Generation Module	user interface
[2000 MITROVIC and SURAWEERA]			Pedagogical agent	SQLT-Web user interface page
[2000 PEYLO]	Lehrstoffspezifisches Wissen	Lernerspezifisches Wissen		Präsentation von Wissen
[2000 ROSATELLI, SELF and THIRY]	Information Agent	Logs	Advising Agent	Interface Agent
[2000 THIBODEAU, BÉLANGER and FRASSON]	Knowledge base	User profile	Learning module	
[2000 THIBODEAU, BÉLANGER and FRASSON]	Knowledge Base	Profile Base		Personal Agents
[2000 UNIVERSITÄT HANNOVER]	Lehrstoffspezifisches Wissen	Benutzermodell	Diagnosewissen; Didaktisches Modell	Lernumgebung
[2000 VIRVOU and MOUNDRIDOU]	Domain description & problems; Domain & Problem Generator or Selector; Problem Solver	Student Model	Instructor Model; Instructor Information Generator; Error Diagnoser; Student Advice Generator	
[2000 VIZCAÍNO, CONTRERAS and FAVELA]	Specific Database	Student Memory		Interface / Collaborative Component
[2000 WIGGINS and TREWIN]	Course Specification	Student Model	Method Library	Graphical & Audio User Interface
[2000 ZOUAQ, FRASSON and ROUANE]			Analyzer; Explanation module; Learning	Perception; Action
[2000 ZOUAQ, FRASSON and ROUANE]	Courses; Exercises; Explanations	Student model	Dialog module	Learner interface
[2001 LUNDGREN-CAYROL, PAQUETTE and MIARA]	Database	Student profile	Student Advisor	
[2002 BAEK, WANG and LEE]	Domain Model	Student Model	Adaptivity Management Module	
[2002 GAMBOA and FRED]	Knowledge Base	User Model	Pedagogical Module/ Adaptation Module	Presentation Module
[2002 HEFFERNAN and KOEDINGER]		Student Model	Tutorial Model	
[2002 NEDIC, NEDIC and MACHOTKA]	Questions Database	Students' Database		
[2002 PRENTZAS, HATZILYGEROUDIS and GAROFALAKIS]	Domain Knowledge	User Modeling Component	Pedagogical Model; Inference Engine	
[2002 SURAWEERA AND ANTONIJA MITROVIC and MITROVIC]	Constraint Based Modeller; Constraints	Student Models	Pedagogical Module; Solution Problems	Interface
[2002 YANG and PATEL]	Expert Module	Student Module	Tutor Module	User Interface
[2003 CROWLEY and MEDVEDEVA]	Expert Module	Student Modeling System	Pedagogic Model	Student Interface
[2003 HARRER]	Expert Module	Student Module1; Student Module2; Student Module3	Tutor Module	UI1; UI2; UI3; Wrapper Interface
[2003 IEEE]	Learning Resources	Learner Records	Coach	Delivery
[2003 IEEE]	LR	R	C	D

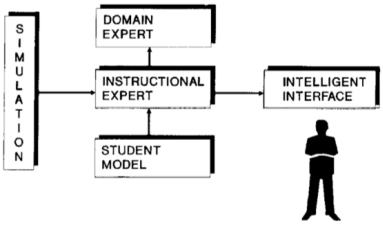
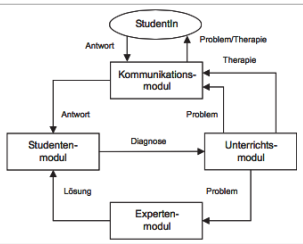
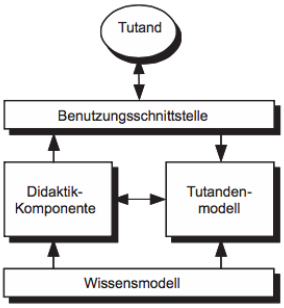
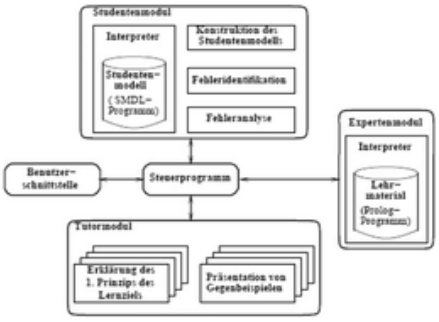
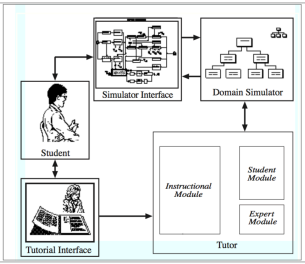
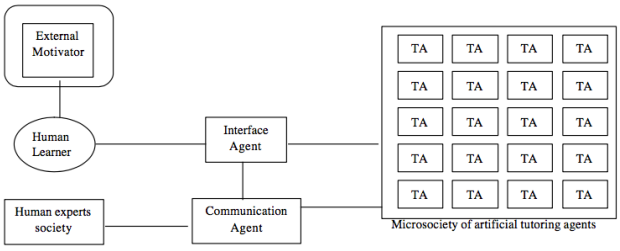
Zitierung	Name vom Domänenwissen	Name vom Lernermodell	Name vom Pädagogikmodul	Name vom Benutzeroberfläche
[2003 MARTENS]	Expert Knowledge Model	Learnermodel	Pedagogical Knowledge Model	User Interface
[2003 MARTENS]	Expert Knowledge Model	Learnermodel	Pedagogical Knowledge Model	User Interface
[2004 GOODKOVSKY]	DOMAIN MODULE			DOMAIN GUI
[2004 HATZILYGEROUDIS and PRENTZAS]	Domain Knowledge	User Modeling Unit	Inference System; Pedagogical Unit	
[2004 JEREMIC, DEVEDZIC and GASEVIC]	Domain module	Student model	Pedagogical module; Expert module	GUI
[2004 LITMAN and SILLIMAN]				
[2004 MELIS and SIEKMANN]	MBase	Student model	Pedagogical rules	presentation engine
[2005 CROWLEY, TSEYTLIN and JUKIC]	Expert Model		Pedagogical Model	Student Interface
[2005 GRAESSER, CHIPMAN and HAYNES]		Assessments Module (2)	Dialogue Management Module (3); Latent Semantic Analysis Utility	Muppet
[2005 KAFKAS, BAYRAM and YARATAN]	Domain Expert Module; Domain Knowledge-Base	Student Model	Pedagogical Module; Student Diagnostic Module; Pedagogical Knowledge	Interface Module
[2005 KARAMPIPERIS and SAMPSON]	Media Space; Domain Model	User Model	Adaptation Model	Educational Content Presenter
[2006 AÏMEUR and ONANA]	Curriculum	Learner model	Planner; Pedagogical strategies; Recommender System	
[2006 CROWLEY and MEDVEDEVA]	Expert model; Case description	Student model	Instructional model	Interface
[2006 DUBOIS, NKAMBOU and HOHMEYER]	domain model	learner profile; learner knowledge model; events log		
[2006 KOJIMA and MIWA]	A dictionary database	Problem data	A casebase	An input-analysis interface
[2006 LEE, CHO and CHOI]	Expert Module	Learner Module	Teacher Module; Diagnosis Module	Interface Module
[2006 LEE, CHO and CHOI]	Contents	Device Profile; Client Profile	Annotation	Layout determination Module; Content Adaption Module
[2006 OERTEL]	Expert Module	User Module	Pedagogical Module; Tutor	User Interface
[2006 OERTEL]	Expertenkomponente	Nutzerkomponente	Pädagogikkomponente	Benutzerschnittstelle
[2007 FUNDA and KADIR]	DOMAIN MODEL	User Model	Pedagogical Model; Instructional Planner; Pedagogic Knowledge Base; Automatic (Dynamic) Course Generation Tool; Adaptive A TUTOR; A Tutor	
[2007 KRAVCIK and GASEVIC]	Domain Model; Context Model	User Model	Adaptation Model; Activity Model	Presentation Specification
[2009 CONATI]	Problem Solver; Domain and planning rules; Problem definitions	Long-term student model; Short-term student model	SE-Coach; EA-Coach; Solution graph(s)	User Interface
[2009 GUNEL and ASLIYAN]	Knowledgebase; Library			User Interface for Experts; User Interface for Students

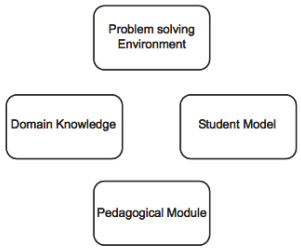
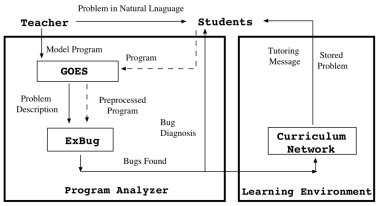
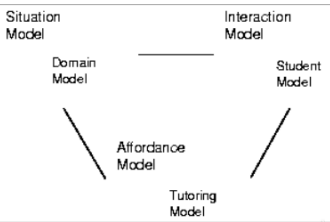
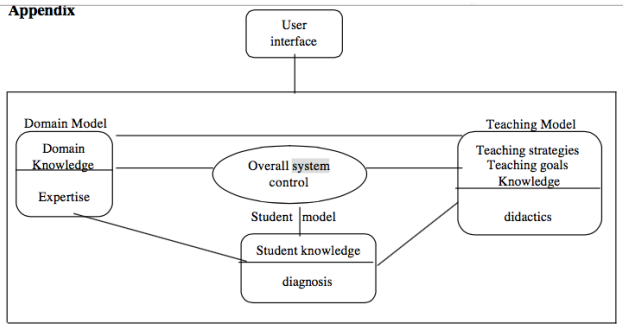
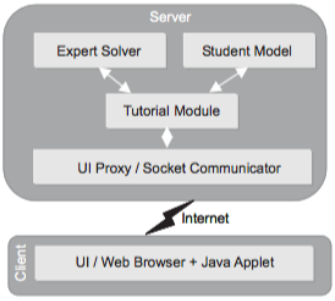
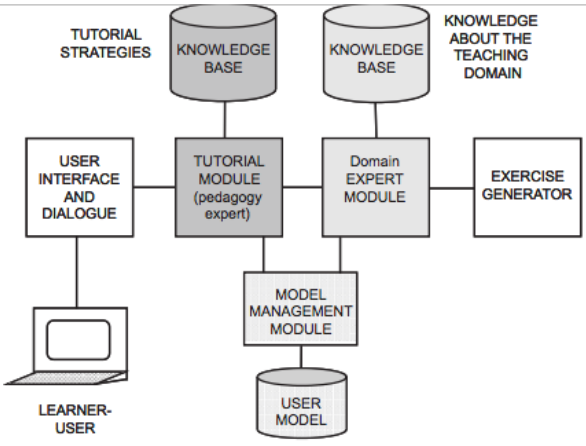
Zitierung	Name vom Domänenwissen	Name vom Lernermodell	Name vom Pädagogikmodul	Name vom Benutzeroberfläche
[2009 MITROVIC, BRENT and SURAWEERA]	Domain Model; Domain Model Manager; Constraint validator; Constraint Generator			Web Interface
[2009 MITROVIC, BRENT and SURAWEERA]	Domain Manager	User Manager; Log Manager; Student Modeller	Pedagogical Module; Diagnostic Module	Interface
[2009a ALEVEN, MCLAREN and SEWALL]		Data Shop	Tutoring Service	Student Interface, Author Interface
[2009b ALEVEN, MCLAREN and SEWALL]	Mathtutor Database	Mathtutor Database, Research Database (DataShop)	Tutoring Service	Student Interface
[2010 CHAKRABORTY, ROY and BASU]	Domain Organization Module; Repository	Student Model		Student's GUI
[2010 GUTIERREZ-SANTOS, MAVRIKIS and MAGOULAS]	Computational Analysis Layer	Student Model	Feedback Layer	Microworld
[2010 NKAMBOU, MIZOGUCHI and BOURDEAU]	Domain Model	Student Model		Interface
[2011 ABU-NASER, AHMED and AL-MASRI]	Expert Module	Student Module	Pedagogical Module	User Interface
[2011 MACIUSZEK and MARTENS]	expert knowledge	learner model	pedagogical process steering	user interface
[2011 MANOUSELIS, DRACHSLER and VUORIKARI]	Learning Resources	User Model	Domain Ontology	Interface Layer
[2011 SOUHAIB, MOHAMED and KADIRI KAMAL EDDINE]	Domain Model	User Model	Adaptation Model	Run-time Layer
[2012 LIN, WANG and CHAO]	Course Database		Teaching Strategy, Feedback Module	User Interface
[2013 AHUJA and SILLE]	EXPERT KNOWLEDGE MODULE	STUDENT MODEL MODULE	TUTORING MODULE	USER INTERFACE MODULE
[2014 GONZALEZ-SANCHEZ, CHAVEZ-ECHEAGARAY and VAN LEHN]	Knowledge Base	Assessor	Pedagogical Module; Task Selector; Step Analyzer	User Interface
[2014 IMRAN]	Rule Repository	Learner Model; Learner Modelling Module	Recommendation Generation Module; Recommendation Display Module	Learning Management System
[2014 PALOMINO, SILVEIRA and NAKAYAMA]	Domain base	Student model	Pedagogical model	

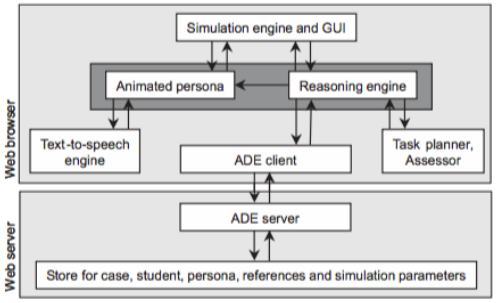
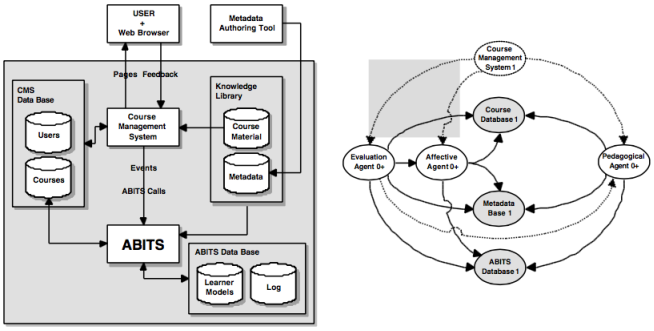
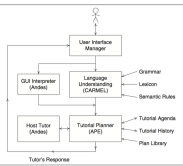
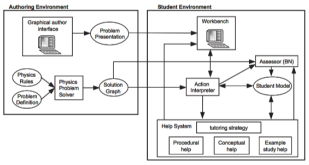
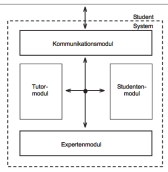
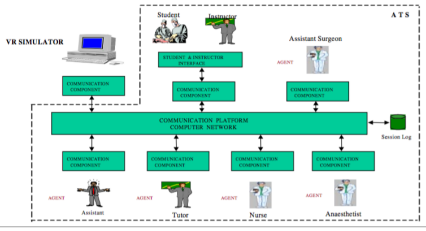
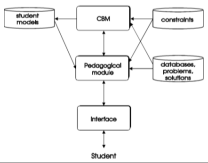
VI.3. Tabelle der ITS-Architekturen

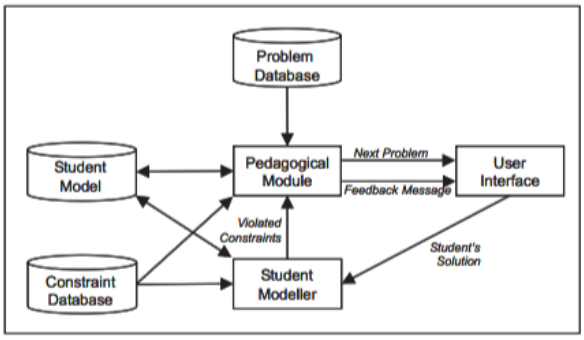
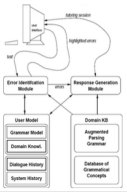
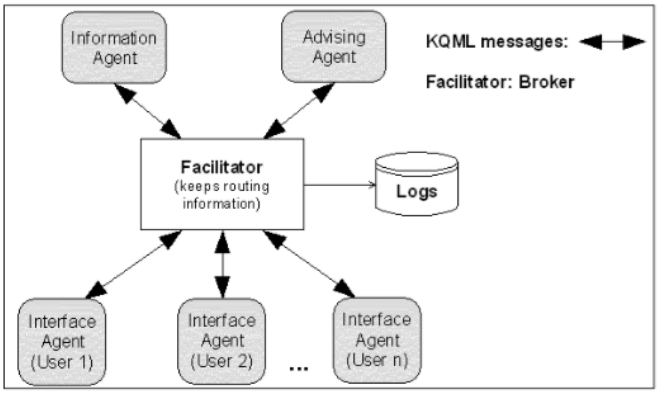
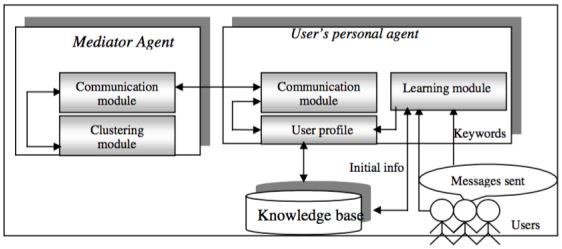
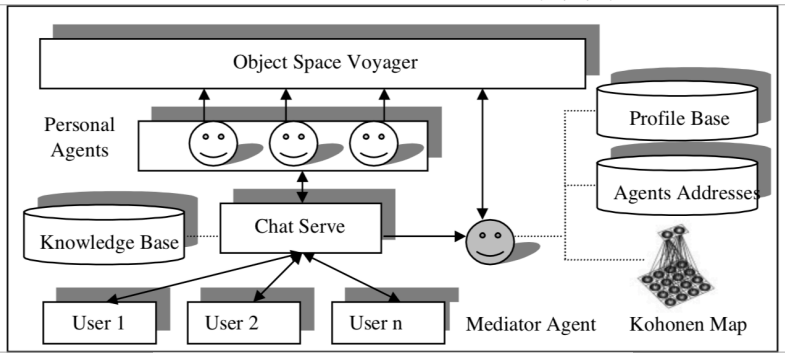
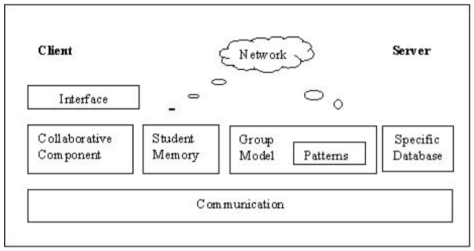
Die Tabelle der ITS-Architekturen hat keinen Anspruch auf Vollständigkeit, versucht aber alle relevanten frei zugänglichen Architekturen vom Anfang bis heute aufzulisten.

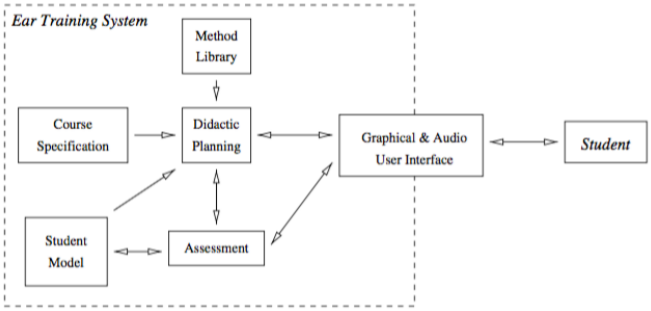
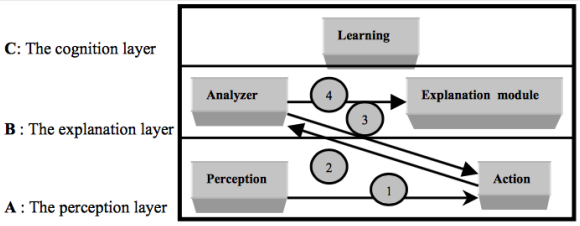
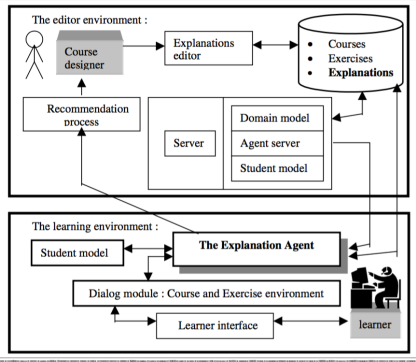
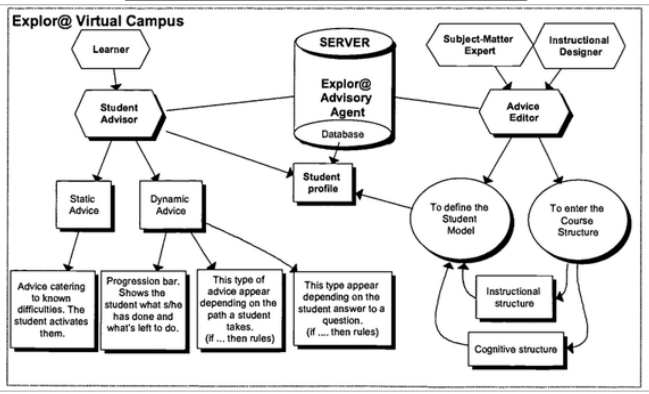
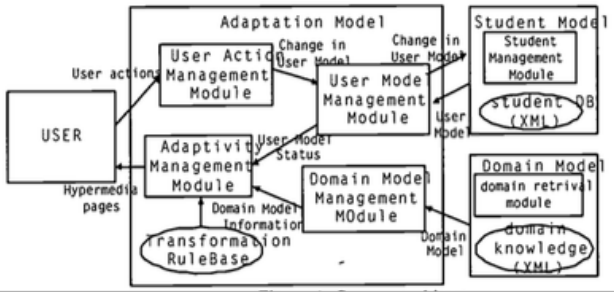
Zitierung von	Bild der Architektur
[1984 CLANCEY]	
[1984 O'SHEA and SELF]	
[1985 ANDERSON, BOYLE and REISER]	
[1990 NWANA]	
[1990 NWANA]	
[1991 ASHLEY and ALEVEN]	

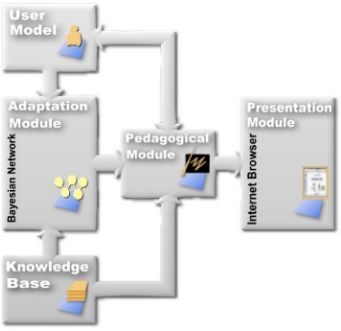
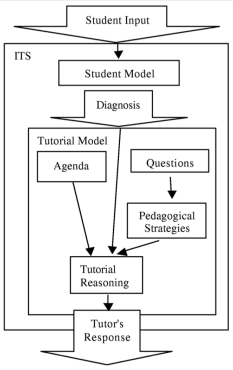
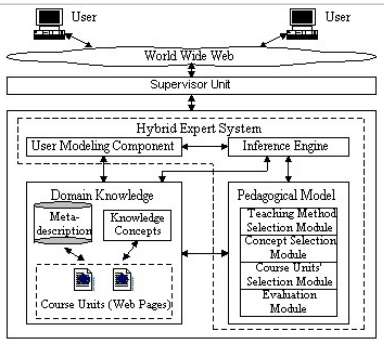
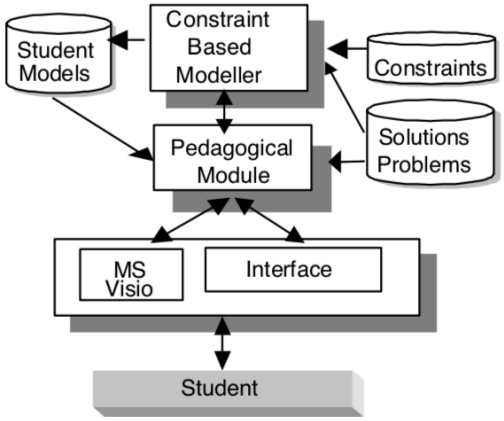
Zitierung von	Bild der Architektur
[1991 BURNS, PARLETT and REDFIELD]	
[1992 LUSTI]	
[1992 PUPPE]	
[1994 IKEDA and MIZOGUCHI]	
[1995 VASANDANI and GOVINDARAJ]	
[1996 DE BARROS COSTA and PERKUSISCH]	

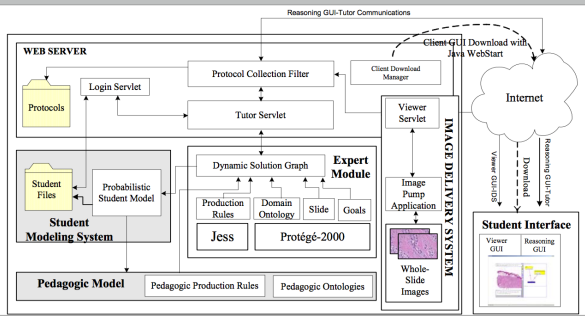
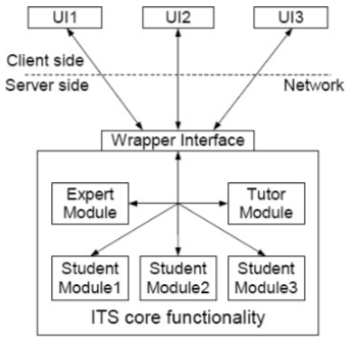
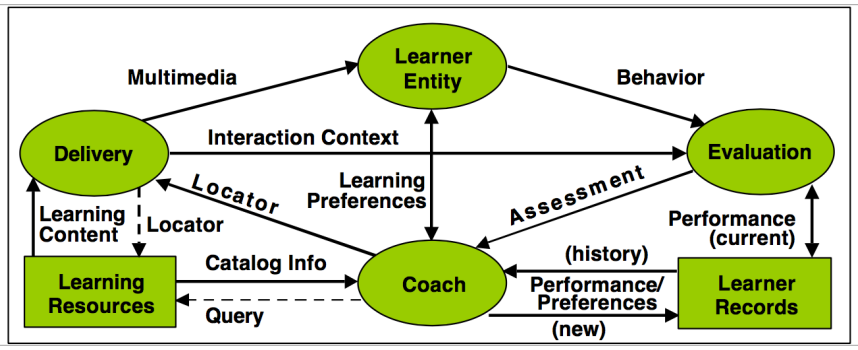
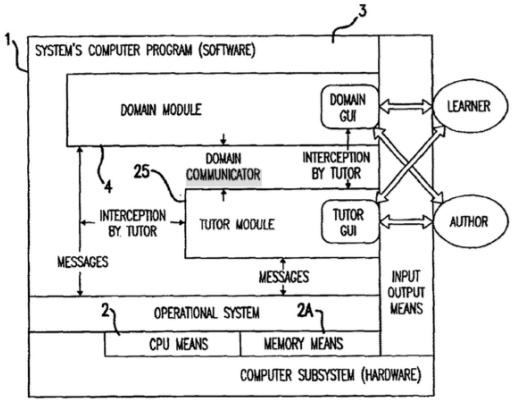
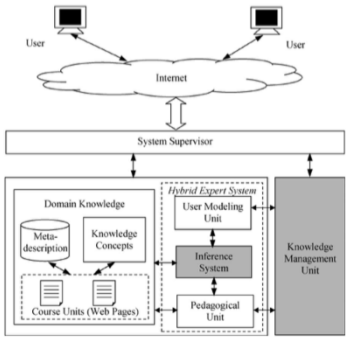
Zitierung von	Bild der Architektur
[1997 CORBETT, KOEDINGER and ANDERSON]	
[1997 SONG, HAHN and TAK]	
[1998 SELF]	
[1998 SIEMER and ANGELIDES]	
[1999 ALPERT, SINGLEY and FAIRWEATHER]	
[1999 LELOUCHE]	

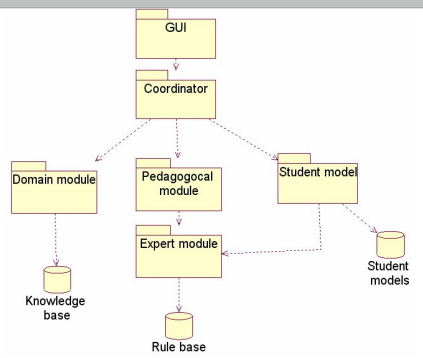
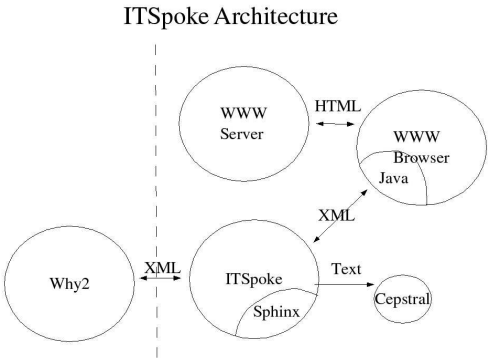
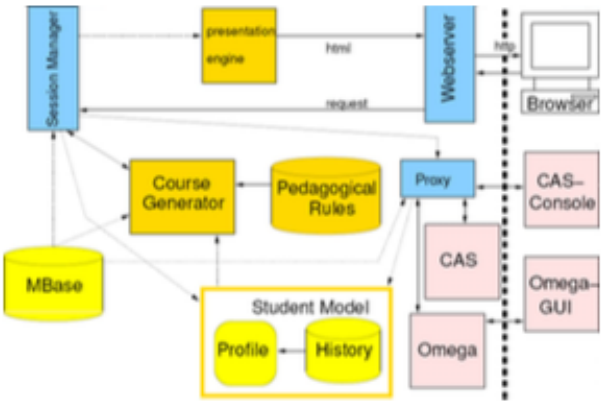
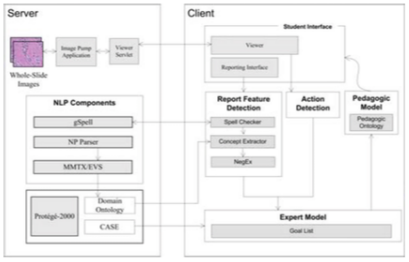
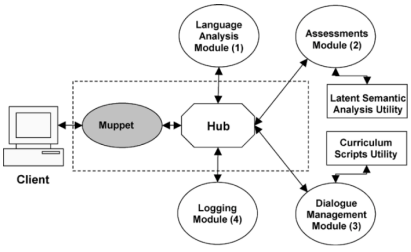
Zitierung von	Bild der Architektur
[1999 SHAW, GANESHAN and JOHNSON]	
[2000 CAPUANO, MARSELLA and SALERNO]	
[2000b FREEDMAN, PENSTEIN and RINGENBERG]	
[2000 GERTNER and VAN LEHN]	
[2000 HARRER]	
[2000 LOS ARCOS, MULLER and FUENTE]	
[2000 MAYO and MITROVIC]	

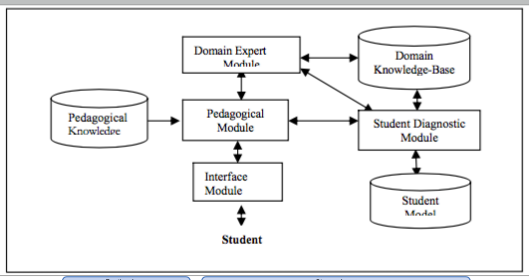
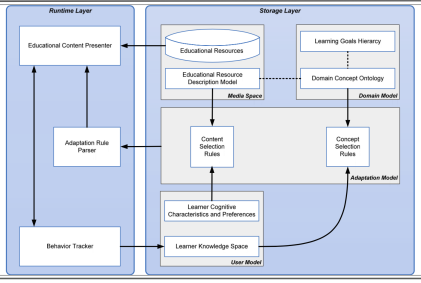
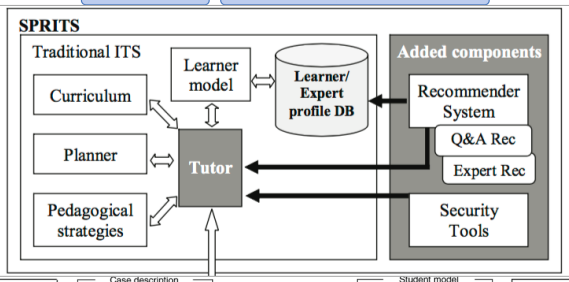
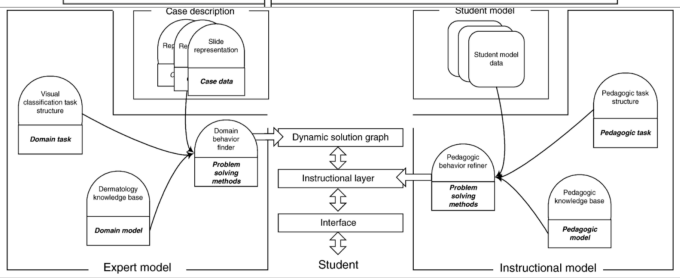
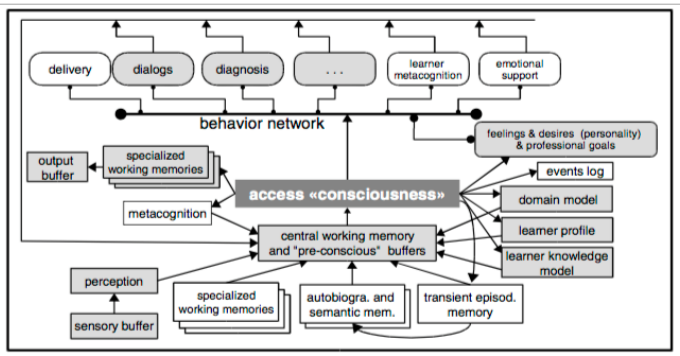
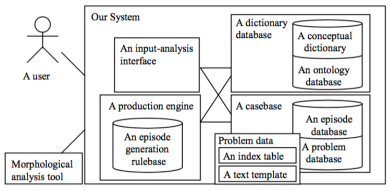
Zitierung von	Bild der Architektur
[2000 MAYO, MITROVIC and MCKENZIE]	
[2000 MICHAUD, MCCOY and PENNINGTON]	
[2000 ROSATELLI, SELF and THIRY]	
[2000 THIBODEAU, BÉLANGER and FRASSON]	
[2000 THIBODEAU, BÉLANGER and FRASSON]	
[2000 VIZCAÍNO, CONTRERAS and FAVELA]	

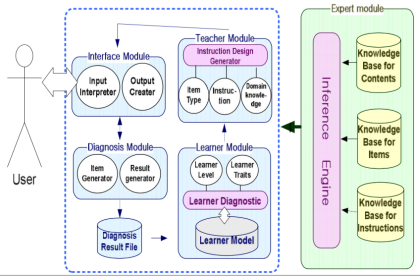
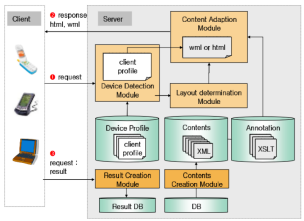
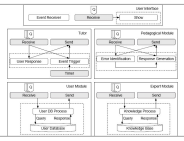
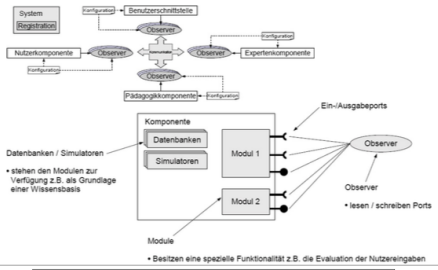
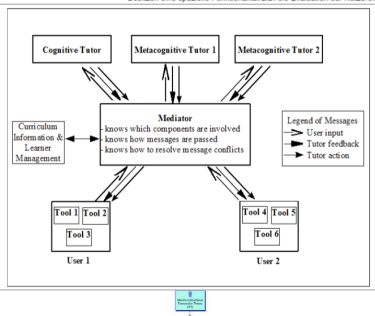
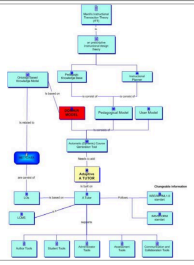
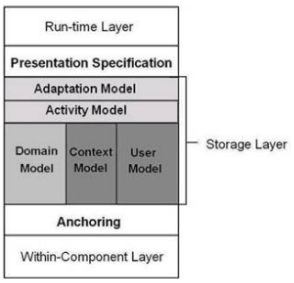
Zitierung von	Bild der Architektur
[2000 WIGGINS and TREWIN]	
[2000 ZOUAQ, FRASSON and ROUANE]	
[2000 ZOUAQ, FRASSON and ROUANE]	
[2001 LUNDGREN-CAYROL, PAQUETTE and MIARA]	
[2002 BAEK, WANG and LEE]	

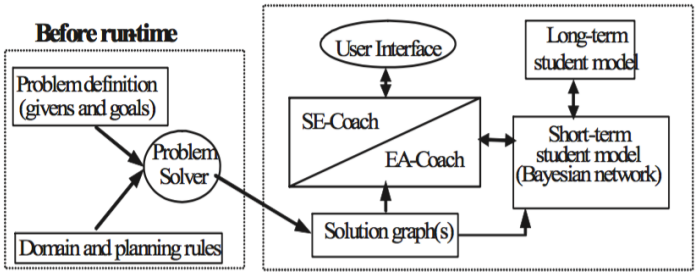
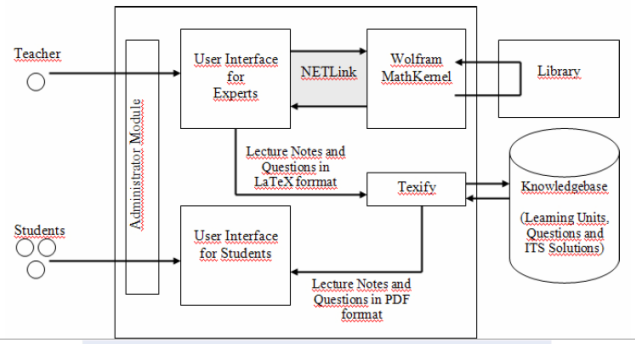
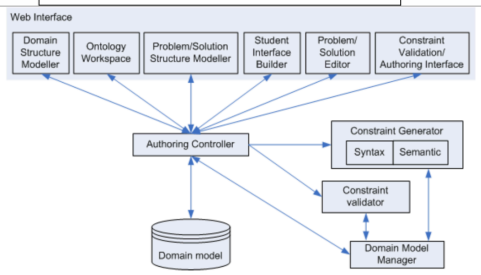
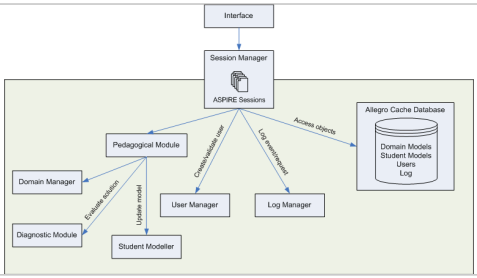
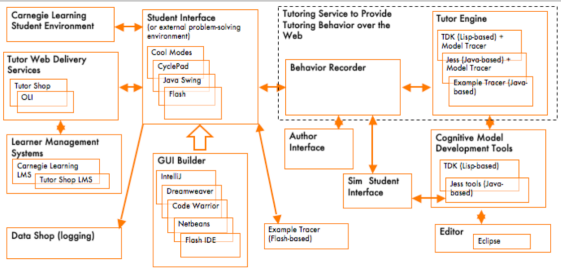
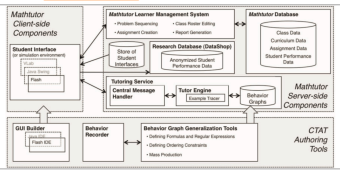
Zitierung von	Bild der Architektur
[2002 GAMBOA and FRED]	
[2002 HEFFERNAN and KOEDINGER]	
[2002 NEDIC, NEDIC and MACHOTKA]	
[2002 PRENTZAS, HATZILYGEROUDIS and GAROFALAKIS]	
[2002 SURAWEERA AND ANTONIJA MITROVIC and MITROVIC]	

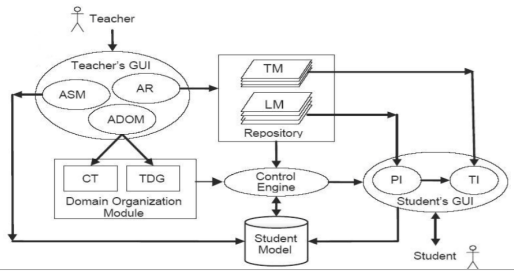
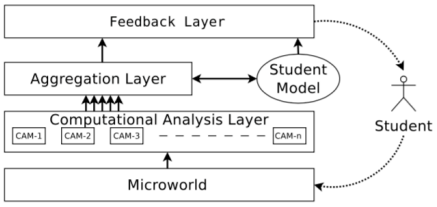
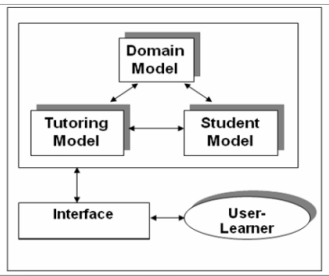
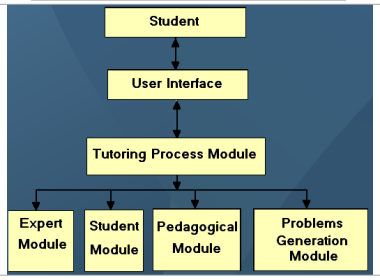
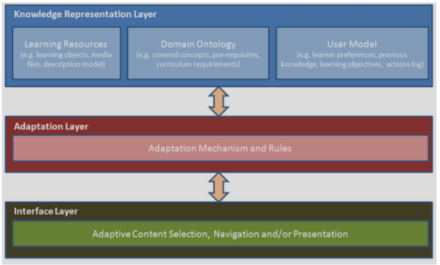
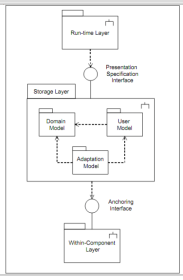
Zitierung von	Bild der Architektur
[2003 CROWLEY and MEDVEDEVA]	
[2003 HARRER]	
[2003 IEEE]	
[2004 GOODKOVSKY]	
[2004 HATZILYGE-ROUDIS and PRENTZAS]	

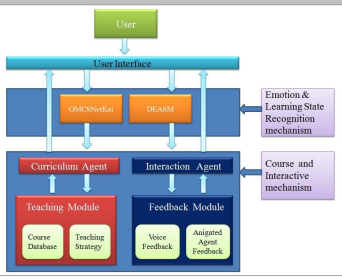
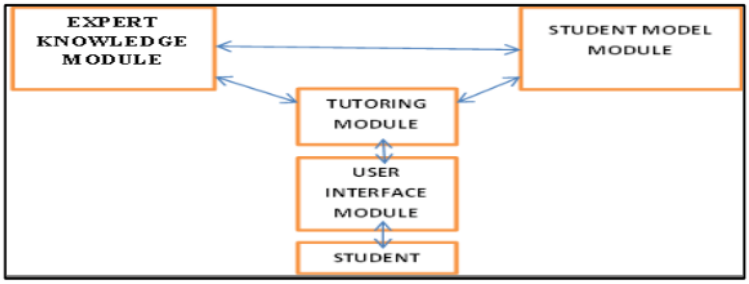
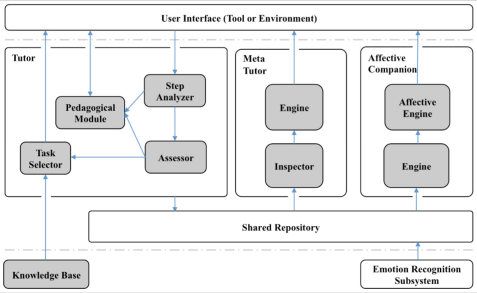
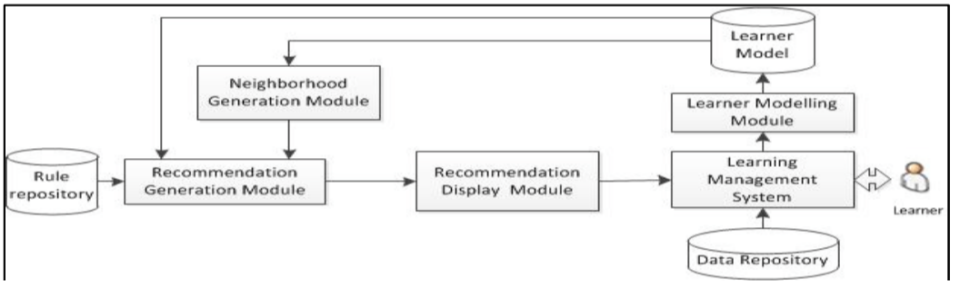
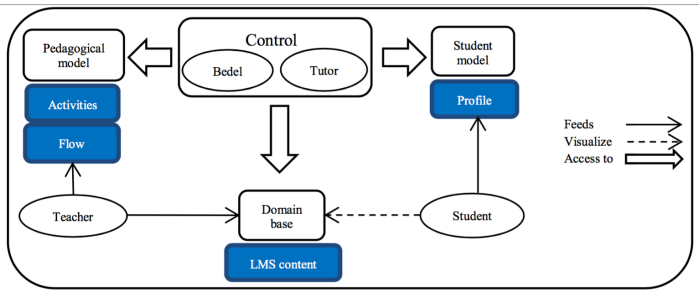
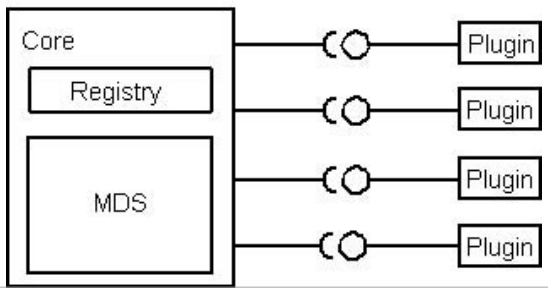
Zitierung von	Bild der Architektur
[2004 JEREMIC, DE-VEDZIC and GASEVIC]	
[2004 LITMAN and SILLIMAN]	
[2004 MELIS and SIEKMANN]	
[2005 CROWLEY, TSEYTLIN and JUKIC]	
[2005 GRAESSER, CHIPMAN and HAY-NES]	

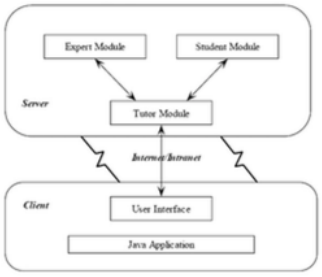
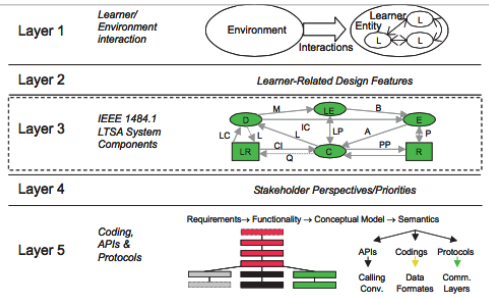
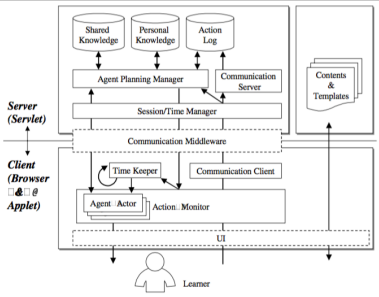
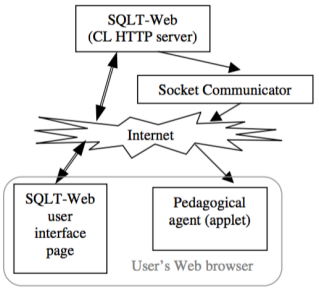
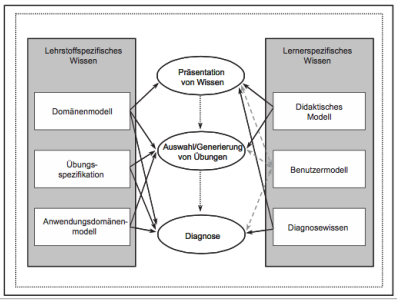
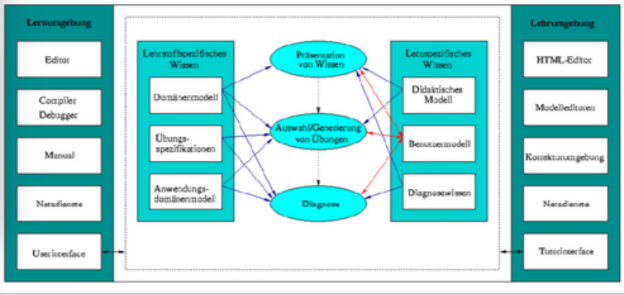
Zitierung von	Bild der Architektur
[2005 KAFKAS, BAYRAM and YARATAN]	
[2005 KARAMPIPERIS and SAMPSON]	
[2006 AÏMEUR and ONANA]	
[2006 CROWLEY and MEDVEDEVA]	
[2006 DUBOIS, NKAMBOU and HOHMEYER]	
[2006 KOJIMA and MIWA]	

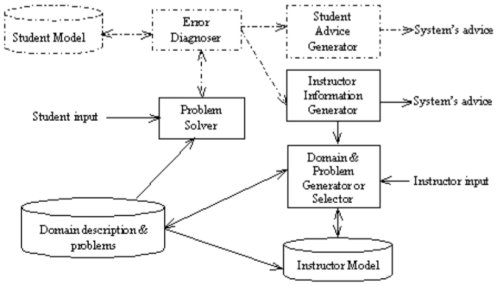
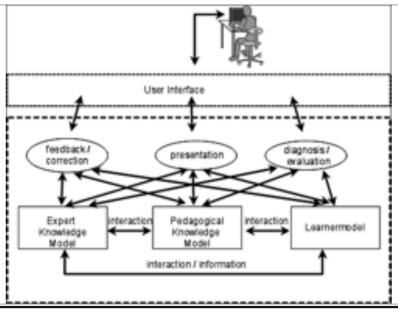
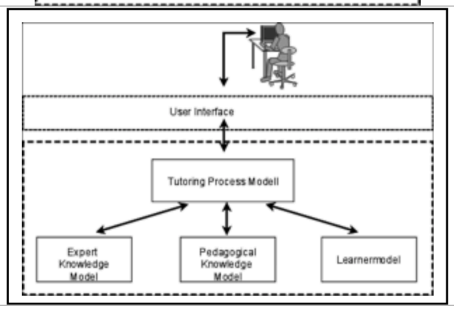
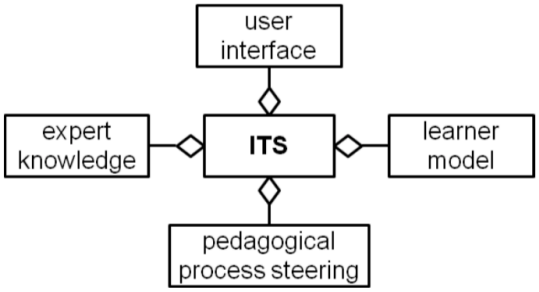
Zitierung von	Bild der Architektur
[2006 LEE, CHO and CHOI]	
[2006 LEE, CHO and CHOI]	
[2006 OERTEL]	
[2006 OERTEL]	
[2006 WALKER, KO-EDINGER and MCLA-REN]	
[2007 FUNDA and KADIR]	
[2007 KRAVCIK and GASEVIC]	

Zitierung von	Bild der Architektur
[2009 CONATI]	
[2009 GUNEL and ASLIYAN]	
[2009 MITROVIC, BRENT and SURA-WEERA]	
[2009 MITROVIC, BRENT and SURA-WEERA]	
[2009a ALEVEN, MCLAREN and SE-WALL]	
[2009b ALEVEN, MCLAREN and SE-WALL]	

Zitierung von	Bild der Architektur
[2010 CHAKRABORTY, ROY and BASU]	
[2010 GUTIERREZ-SANTOS, MAVRIKIS and MAGOULAS]	 **Fig. 1.** Layered design and evaluation of intelligent support
[2010 NKAMBOU, MIZOGUCHI and BOURDEAU]	
[2011 ABU-NASER, AHMED and AL-MASRI]	
[2011 MANOUSELIS, DRACHSLER and VUORIKARI]	
[2011 SOUHAIB, MOHAMED and KADIRI KAMAL EDDINE]	

Zitierung von	Bild der Architektur
[2012 LIN, WANG and CHAO]	
[2013 AHUJA and SILLE]	
[2014 GONZALEZ-SANCHEZ, CHAVEZ-ECHEAGARAY and VAN LEHN]	
[2014 IMRAN]	
[2014 PALOMINO, SILVEIRA and NAKAYAMA]	
[2006 OERTEL]	

Zitierung von	Bild der Architektur
[2002 YANG and PATEL]	
[2003 IEEE]	
[2000 MATSUURA, OGATA and YANO]	
[2000 MITROVIC and SURAWEERA]	
[2000 PEYLO]	
[2000 UNIVERSITÄT HANNOVER]	

Zitierung von	Bild der Architektur
[2000 VIRVOU and MOUNDRIDOU]	
[2003 MARTENS]	
[2003 MARTENS]	
[2011 MACIUSZEK and MARTENS]	

VI.4. Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die beigefügte Dissertation selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel genutzt habe. Alle wörtlich oder inhaltlich übernommenen Stellen habe ich als solche gekennzeichnet.

Ich versichere außerdem, dass ich die beigefügte Dissertation nur in diesem und keinem anderen Promotionsverfahren eingereicht habe und, dass diesem Promotionsverfahren keine endgültig gescheiterten Promotionsverfahren vorausgegangen sind.

20. November 2020
Admannshagen

_____	_____
Datum, Ort	Unterschrift

VI.5. Danksagung

Einen unglaublichen Dank geht an meine Doktormutter Frau Prof. Dr.-Ing. Alke Martens für ihre außerordentlich kompetente und zusätzlich ihr immerwährende überaus freundliche Unterstützung.

Vielen Dank an meine Ehefrau für ihren immerwährenden Beistand und ihr Verständnis. Natürlich danke ich auch meiner Tochter, die es zu jeder Zeit schafft mir ein Lächeln zu entlocken und mir die Motivation gibt ein Vorbild zu sein.

Die Kapitel III., IV., V. und die Verteidigung der Promotion wurden im Rahmen des Forschungsprojektes „DigiCare“ angefertigt und durch Mittel des Europäischen Sozialfonds (ESF) finanziert (ESF/14-BM-A55-0018/19). Die vorliegende Arbeit ist Bestandteil des Qualifikationsprogrammes „Förderung von Nachwuchswissenschaftlern in exzellenten Forschungsverbünden - Exzellenzforschungsprogramm des Landes Mecklenburg-Vorpommern“.

VI.6. Eigene Veröffentlichungen

- [2016 GRAF VON MALOTKY and MARTENS]: 2016; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; Framework for intelligent teaching and training systems - a study of systems; 13th International Conference Cognition and Exploratory Learning in the Digital Age (CELDA), ERIC number ED571423, ISBN 978-989-8533-55-5.
- [2017a GRAF VON MALOTKY, NAGORSNICK and MARTENS]: 2017a; Nikolaj Troels GRAF VON MALOTKY, Marian NAGORSNICK, Alke MARTENS; Hyperspectral Wound Diagnostic – Analysis of Clinical Practice and Diagnosis Support; 62. Jahrestagung Deutsche Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e. V. (GMDS), publisher gms.
- [2017b GRAF VON MALOTKY, NAGORSNICK and MARTENS]: 2017b; Nikolaj Troels GRAF VON MALOTKY, Marian NAGORSNICK, Alke MARTENS; Hyperspectral Imaging and Clinical Practice – Approach to Integrate a New Diagnostic Tool; 51. Jahrestagung der BIOMEDIZINISCHEN TECHNIK und Dreiländertagung der MEDIZINISCHEN PHYSIK (BMTMedPhys), Deutsche Gesellschaft für Biomedizinische Technik.
- [2017 GRAF VON MALOTKY, NAGORSNICK and NICOLAY]: 2017; Nikolaj Troels GRAF VON MALOTKY, Marian NAGORSNICK, Robin NICOLAY; Bringing Hyperspectral Imaging into Clinical Practice with Computer Science Methods; Biomedizinische Technik / Biomedical Engineering Journal (BMT), ISSN 1862-278X, OCLC 163426070.
- [2017a GRAF VON MALOTKY, NICOLAY and MARTENS]: 2017a; Nikolaj Troels GRAF VON MALOTKY, Robin NICOLAY, Alke MARTENS; Synthesis of pedagogical annotations; 9th annual International Conference on Education and New Learning Technologies (EDULEARN), EDULEARN17 Proceedings, ISBN 978-84-697-3777-4, ISSN 2340-1117, DOI 10.21125/edulearn.2017, pages 3655-3661.
- [2017b GRAF VON MALOTKY, NICOLAY and MARTENS]: 2017b; Nikolaj Troels GRAF VON MALOTKY, Robin NICOLAY, Alke MARTENS; Centralizing the Teaching Process in Intelligent Tutoring System Architectures; 19th International Conference on Advanced Learning Technologies (ICALT), Open Science Index, Educational and Pedagogical Sciences, volume 11, number 10, ISNI 0000000091950263.
- [2017a MARTENS, NICOLAY and GRAF VON MALOTKY]: 2017a; Alke MARTENS, Robin NICOLAY, Nikolaj Troels GRAF VON MALOTKY; Integration neuer diagnostischer Verfahren in den klinischen Alltag; Symposium Hyperspektrale Bildgebung in der Medizin (HSI), https://tu-dresden.de/ing/elektrotechnik/ibmt/ressourcen/dateien/ibmt/tagungen/tagungen-2017/hsi-symposium/2017_HSI_Symposium.pdf.
- [2017b MARTENS, NICOLAY and GRAF VON MALOTKY]: 2017b; Alke MARTENS, Robin NICOLAY, Nikolaj Troels GRAF VON MALOTKY; Hyperspectral imaging and clinical practice – approach to integrate a new diagnostic tool; forschungscamp, Universität Rostock.
- [2017 NICOLAY, GRAF VON MALOTKY and AUGE]: 2017; Robin NICOLAY, Nikolaj Troels GRAF VON MALOTKY, Tanja AUGE; Autonomous semantic structuring of lecture topics Synthesis of knowledge models; 9th International Conference on Computer Supported Education CSEDU, volume 2, pages 349-355, SCITEPRESS, DOI 10.5220/0006367903490355.
- [2019b GRAF VON MALOTKY and MARTENS]: 2019b; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; Analyzing the usage of the classical ITS software architecture and refining it; 15th international conference on intelligent tutoring systems (ITS2019), pages 40-46, DOI 10.1007/978-3-030-22244-4_6, Springer, Cham, Online ISBN 978-3-030-22244-4, Print ISBN 978-3-030-22243-7.

- [2019 WASSMANN, GRAF VON MALOTKY and MARTENS]: 2019; Ingolf WASSMANN, Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; Train4U - Mobile Sport Diagnostic Expert System for User-Adaptive Training; Proceedings of the 12th International Symposium on Computer Science in Sport (IACSS), Advances in Intelligent Systems and Computing, volume 1028, Springer, Cham, pages 77-85.
- [2019a GRAF VON MALOTKY and MARTENS]: 2019a; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; A comparable foundation for ITS research loop and subfields; 11th annual International Conference on Education and New Learning Technologies (EDULEARN), pages 1402-1409, DOI 10.21125/edulearn.2019.0425.
- [2019 VAWTER, GRAF VON MALOTKY and MARTENS]: 2019; Laura VAWTER, Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; The Implications of ITS Research on CALL Software; EDULEARN19, the 11th annual International Conference on Education and New Learning Technologies, pages, 1562-1568, DOI 10.21125/edulearn.2019.0463.
- [2020a GRAF VON MALOTKY and MARTENS]: 2020a; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; General ITS software architecture and framework; 16th International Conference on Intelligent Tutoring Systems (ITS2020), Springer.
- [2020b GRAF VON MALOTKY and MARTENS]: 2020b; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; DigiCare - Ein intelligentes Lehr-/Lernsystem für Gesundheitsberufe; 18. Fachtagung Bildungstechnologien (DELFI), Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2020.

VI.7. Bibliographie

- [1995 ABOWD, ALLEN and GARLAN]: 1995; Gregory D. ABOWD, Robert ALLEN, David GARLAN; Formalizing style to understand descriptions of software architecture; ACM Transactions on Software Engineering and Methodology (TOSEM) 4, number 4, pages 319-364.
- [2011 ABU-NASER, AHMED and AL-MASRI]: 2011; S. ABU-NASER, A. AHMED, N. AL-MASRI, A. DEEB, E. MOSHTAHA, M. ABU-LAMDY; An Intelligent Tutoring System for Learning Java Objects; International Journal of Artificial Intelligence & Applications (IJAIA), volume 2, number 2.
- [2002 ADL]: 2002; ADL; ADL Technical Team: SCORM Specification 1.2; http://www.adl-net.org/adl-research/scorm/ scorm-1-2/, 2002.
- [2013 AHUJA and SILLE]: 2013; Neelu Jyothi AHUJA, Roohl SILLE; A Critical Review of Development of Intelligent Tutoring Systems: Retrospect, Present and Prospect; IJCSI International Journal of Computer Science issues, volume 10, issue 4, number 2, ISSN (Print): 1694-0814, ISSN (Online) 1694-0784.
- [2006 AÏMEUR and ONANA]: 2006; Esma AÏMEUR, Flavien Serge Mani ONANA; Sprits: Secure pedagogical resources in intelligent tutoring systems; International Conference on Intelligent Tutoring Systems, pages 237-247, Springer Verlag, Berlin, Heidelberg.
- [2006 ALEVEN, MCLAREN and SEWALL]: 2006; Vincent ALEVEN, Bruce M. MCLAREN, Jonathan SEWALL, Kenneth R. KOEDINGER; The cognitive tutor authoring tools (CTAT): preliminary evaluation of efficiency gains; International conference on Intelligent Tutoring Systems, pages 61-70, Springer Verlag, Berlin, Heidelberg.
- [2009a ALEVEN, MCLAREN and SEWALL]: 2009a; Vincent ALEVEN, Bruce M. MCLAREN, Jonathan SEWALL, Kenneth R. KOEDINGER; A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors; International Journal of Artificial Intelligence in Education 19, IOS Press, pages 105-154.
- [2009b ALEVEN, MCLAREN and SEWALL]: 2009b; Vincent ALEVEN, Bruce M. MCLAREN, Jonathan SEWALL; Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning; IEEE Transactions on Learning Technologies 2, number 2, pages 64-78.
- [2016 ALEVEN, MCLAREN and SEWALL]: 2016; Vincent ALEVEN, Bruce M. MCLAREN, Jonathan SEWALL, Martin VAN VELSEN, Octav POPESCU, Sandra DEMI, Michael RINGENBERG, Kenneth R. KOEDINGER; Example-Tracing tutors: Intelligent tutor development for non-programmers; International Journal of Artificial Intelligence in Education 26, number 1, pages 224-269..
- [2010 ALI, DEHGHAN and GHOLAMPOUR]: 2010; A. Pouyan ALI, Hossein DEHGHAN, Javad GHOLAMPOUR; An agent based multilayered architecture for e-learning system; Conference The Second International Conference on E-Learning and E-Teaching (ICELET 2010), Publisher IEEE, pages 22-26, ISBN 1424490111.
- [1999 ALPERT, SINGLEY and FAIRWEATHER]: 1999; Sherman R. ALPERT, Mark K. SINGLEY, Peter G. FAIRWEATHER; Deploying intelligent tutors on the web: An architecture and an example; International Journal of Artificial Intelligence in Education (IJAIED) 10, number 2, pages 183-197, HAL hal-00197339.
- [2020 AMAZON]: 2020; AMAZON; Alexa; https://www.amazon.de/b?ie=UTF8&node=12775495031.
- [2017 AMBOSS]: 2017; AMBOSS; A learning program for the students of human medicine for the state examination; https://amboss.miamed.de.

- [1985 ANDERSON, BOYLE and REISER]: 1985; J. R. ANDERSON, D. G. BOYLE, B. REISER; Intelligent Tutoring Systems; Science 228, pages 456-462.
- [1985 ANDERSON, BOYLE and YOST]: 1985; J. R. ANDERSON, C. F. BOYLE, G. YOST; The Geometry Tutor; Proceedings of the 9th International Joint Conference on Artificial Intelligence, pages 1-7, San Francisco: Morgan Kaufmann.
- [1990 ANDERSON]: 1990; J. R. ANDERSON; The adaptive character of thought; Book, Lawrence Erlbaum Associates, Hillsdale.
- [2001 ANDERSON]: 2001; John R. ANDERSON; Kognitive Psychologie; Book, Spektrum Akademischer Verlag, volume 3.
- [2018 APPLE INCORPORATED]: 2018; APPLE INCORPORATED; Model-View-Controller; https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.
- [2020 APPLE INCORPORATED]: 2020; APPLE INCORPORATED; Siri; https://www.apple.com/de/siri/.
- [1991 ASHLEY and ALEVEN]: 1991; Kevin D. ASHLEY, Vincent ALEVEN; Toward an intelligent tutoring system for teaching law students to argue with cases; Proceedings of the 3rd international conference on Artificial intelligence and law, publisher ACM, pages 42-52, ISBN 089791399X.
- [1974 AUSUBEL, NOVAK and HANESIAN]: 1974; David P. AUSUBEL, Joseph D. NOVAK, Helen HANESIAN; Psychologie des Unterrichts; Book, Beltz, Harvard.
- [2011 BACHMANN, BASS and GARLAN]: 2011; Felix BACHMANN, Len BASS, Davin GARLAN, James IVERS, Reed LITTLE, Paulo MERSON, Robert NORD, Judith STAFFORD; Documenting Software Architectures: Views and Beyond; Addison-Wesley Professional, book.
- [2017 BACK, VON MALOTKY and SOSTMANN]: 2017; David Alexander BACK, Jennifer VON MALOTKY, Kair SOSTMANN, Robert HUBE, Harm PETERS, Eike HOFF; Superior gain in knowledge by podcasts versus text-based learning in teaching orthopedics: A randomized controlled trial; Journal of surgical education 74.1, pages 154-160.
- [2002 BAEK, WANG and LEE]: 2002; Yeongtae BAEK, Changjong WANG, Sehoon LEE; Adaptive Hypermedia Educational System Based on XML Technologies.; ED-MEDIA 2002 World Conference on Educational Multimedia, Hypermedia & Telecommunications, Proceedings 14th, Denver, Colorado, see IR 021 687.
- [1977 BANDURA]: 1977; Albert BANDURA; Social learning theory; Englewood Cliffs, NJ: Prantice Hall, Peer involvement in bullying: insight and challenges for intervention. Journal of Adolescence, number 22.
- [2003 BASS, CLEMENTS and KAZMAN]: 2003; Len BASS, Paul CLEMENTS, Rick KAZMAN; Software architecture in practice; Addison-Wesley Professional, Book (2nd edition), Addison-Wesley, page 27.
- [2005 BAUSERMAN, CASSADY and SMITH]: 2005; Kathryn L. BAUSERMAN, Jerrell C. CASSADY, Lawrence L. SMITH, James C. STROUD; Kindergarten literacy achievement: The effects of the PLATO integrated learning system; Literacy Research and Instruction 44, number 4, pages 49-60.
- [2010 BEAL, ARROYO and COHEN]: 2010; Carole R. BEAL, Ivon ARROYO, Paul R. COHEN, Beverly P. WOOLF; Evaluation of AnimalWatch: An intelligent tutoring system for arithmetic and fractions; Journal of Interactive Online Learning, volume 9, issue 1, pages 64-77, ISSN 1541-4914.

- [1992 BHANSALI and NII]: 1992; Sanjay BHANSALI, H. Penny NII; Software design by reusing architectures; Knowledge-Based Software Engineering Conference, Proceedings of the Seventh, IEEE, pages 100-109.
- [2001 BIGGS]: 2001; John BIGGS; The reflective institution: Assuring and enhancing the quality of teaching and learning; Higher education 41, number 3, pages 221-238.
- [2014 BIGGS and COLLIS]: 2014; John B. BIGGS, Kevin F. COLLIS; Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome); Academic Press.
- [1995 BOASSON]: 1995; Maarten BOASSON; The artistry of software architecture; IEEE Software 12, number 6, page 13.
- [1999 BOOCH, RUMBAUGH and JACOBSON]: 1999; G. BOOCH, J. RUMBAUGH, I. JACOBSON; Rational Unified Process; Unified Modeling Language–Guia do Usuário. Apêndice C (1999): 442-448.
- [2010 BRAUN]: 2010; Torsten BRAUN; Geschichte und Entwicklung des Internets; Journal Informatik-Spektrum, volume 33, issue 2, pages 201-207, ISSN 0170-6012.
- [2004 BREMER]: 2004; Claudia BREMER; Szenarien mediengestützten Lehrens und Lernens in der Hochschule; Journal Alice im Wunderland-E-Learning an deutschen Hochschulen, Vision und Wirklichkeit, Bielefeld Bertelsmann, http://www.bremer.cx/paper23/paper_bremer_alicebuch.pdf.
- [2006 BROWN and ESKENAZI]: 2006; Jonathan BROWN, Maxine ESKENAZI; Using simulated students for the assessment of authentic document retrieval; International Conference on Intelligent Tutoring Systems, pages 685-688, Springer.
- [1997 BRUSILOVSKY]: 1997; Peter BRUSILOVSKY; Brusilovsky, Peter, Steven Ritter, and Elmar Schwarz. "Distributed intelligent tutoring on the Web; Artificial Intelligence in Education: Knowledge and Media in Learning Systems. IOS, Amsterdam 482, page 489.
- [1998a BRUSILOVSKY]: 1998a; Peter BRUSILOVSKY; Adaptive Educational Systems on the World-Wide-Web: A Review of Available Technologies; Conference Proceedings of Workshop WWW-Based Tutoring at 4th International Conference on Intelligent Tutoring Systems, San Antonio, TX.
- [1998b BRUSILOVSKY]: 1998b; Peter BRUSILOVSKY; Methods and techniques of adaptive hypermedia; Adaptive hypertext and hypermedia, pages 1-43, Springer.
- [1991 BURNS, PARLETT and REDFIELD]: 1991; Hugh BURNS, James W. PARLETT, Carol L. REDFIELD; Intelligent tutoring systems: Evolution in design; Lawrence Erlbaum Associates, Hillsdale, NJ.
- [2000 CAPUANO, MARSELLA and SALERNO]: 2000; Nicola CAPUANO, Marco MARSELLA, Saverio SALERNO; ABITS: An agent based Intelligent Tutoring System for distance learning; In Proceedings of the International Workshop on Adaptive and Intelligent Web-Based Education Systems, ITS.
- [1970 CARBONELL]: 1970; Jaime R. CARBONELL; AI in CAI: An artificial-intelligence approach to computer-assisted instruction; Man-Machine Systems, IEEE Transactions on, volume 11, number 4, pages 190-202, ISSN 0536-1540, DOI 10.1109/TMMS.1970.299942.
- [2018 CARNEGIE MELLON UNIVERSITY]: 2018; CARNEGIE MELLON UNIVERSITY; Cognitive tutor authoring tools; http://ctat.pact.cs.cmu.edu.
- [2006 CHA, KIM and PARK]: 2006; Hyun Jin CHA, Yong Se KIM, Seon Hee PARK, Tae Bok YOON, Young Mo JUNG, Jee-Hyong LEE; Learning styles diagnosis based on user interface behaviors for the customization of learning interfaces in an intelligent tutoring

system; International Conference on Intelligent Tutoring Systems, pages 513-524, Springer, Berlin, Heidelberg.

- [2010 CHAKRABORTY, ROY and BASU]: 2010; Sunandan CHAKRABORTY, Devshri ROY, Anupam BASU; Development of Knowledge Based Intelligent Tutoring System; TMRF e-Book Advanced Knowledge Based Systems: Model, Applications & Research Eds. Sajja & Akerkar, volume 1, pages 74-100.
- [2010 CHI and ROY]: 2010; Michelene T. H. CHI, Marguerite ROY; How adaptive is an expert human tutor?; International Conference on Intelligent Tutoring Systems, Publisher Springer, pages 401-412.
- [2014 CHI, JORDAN and VAN LEHN]: 2014; Min CHI, Pamela JORDAN, Kurt VAN LEHN; When is tutorial dialogue more effective than step-based tutoring?; Conference Intelligent Tutoring Systems, Springer International Publishing.
- [1984 CLANCEY]: 1984; William J. CLANCEY; Methodology for building an intelligent tutoring system; Methods and tactics in cognitive science.
- [1986 CLANCEY]: 1986; William J. CLANCEY; From GUIDON to NEOMYCIN and HERACLES in twenty short lessons; AI Magazine, volume 7, number 3, pages 40,ISSN 0738-4602.
- [2002 CLEMENTS]: 2002; Paul C. CLEMENTS; Software architecture in practice; Diss. Software Engineering Institute.
- [2018 COLOSSAL ORDER LTD. and PARADOX INTERACTIVE]: 2018; COLOSSAL ORDER LTD., PARADOX INTERACTIVE; Cities Skylines; https://www.paradoxplaza.com/cities-skylines/CSCS00GSK-MASTER.html.
- [2018 COMPUTERHISTORY.ORG]: 2018; COMPUTERHISTORY.ORG; LPG-30 image; http://s7.computerhistory.org/is/image/CHM/x14.81p-03-01?$re-story-hero$.
- [2002 CONATI, GERTNER and VAN LEHN]: 2002; Cristina CONATI, Abigail GERTNER, Kurt VAN LEHN; Using Bayesian networks to manage uncertainty in student modeling; User modeling and user-adapted interaction 12 number 3, pages 371-417.
- [2009 CONATI]: 2009; Cristina CONATI; Intelligent Tutoring Systems: New Challenges and directions; Conference Proceeding (IJCAI), volume 9.
- [2018 COOLMATH4KIDS]: 2018; COOLMATH4KIDS; Math + Games for Kids, Teachers & Parents; http://www.coolmath4kids.com.
- [1993 CORBETT and ANDERSON]: 1993; Albert T. CORBETT, John R. ANDERSON; Student modeling in an intelligent programming tutor; Cognitive models and intelligent environments for learning programming, pages 135-144, Springer Verlag, Berlin, Heidelberg.
- [1995 CORBETT, ANDERSON and O'BRIEN]: 1995; Albert T. CORBETT, John R. ANDERSON, Alison T. O'BRIEN; Student modeling in the ACT programming tutor; Cognitively diagnostic assessment, pages 19-41.
- [1997 CORBETT and BHATNAGAR]: 1997; Albert T. CORBETT, Akshat BHATNAGAR; Student modeling in the ACT programming tutor: Adjusting a procedural learning model with declarative knowledge; User modeling, pages 243-254, Springer Verlag, Wien.
- [1997 CORBETT, KOEDINGER and ANDERSON]: 1997; Albert T. CORBETT, Kenneth R. KOEDINGER, John R. ANDERSON; Intelligent Tutoring Systems; Handbook of Human-Computer Interaction, Second, Completely Revised Edition, Elsevier Science B. V., second edition, chapter 37, pages 849–874.
- [1994 CRISPEN and STUCKEY]: 1994; Robert G. CRISPEN, Lynn D. STUCKEY; Structural model: Architecture for software designers; Proceedings of the conference on TRI-Ada'94, pages 272-281, ACM.

- [1959 CROWDER and GALANTER]: 1959; Norman A. CROWDER, E. GALANTER; Automatic tutoring by means of intrinsic programming; Wiley, pages 109-116, New York.
- [2003 CROWLEY and MEDVEDEVA]: 2003; Rebecca CROWLEY, Olga MEDVEDEVA; SlideTutor: A model-tracing Intelligent Tutoring System for teaching microscopic; Artificial Intelligence in Education: Shaping the Future of Learning Through Intelligent Technologies, volume 97, pages 157 ISBN/ISSN 4274906000.
- [2005 CROWLEY, TSEYTLIN and JUKIC]: 2005; Rebecca S. CROWLEY, Eugene TSEYTLIN, Drazen JUKIC; ReportTutor – an intelligent tutoring system that uses a natural language interface; American Medical Informatics Association, AMIA Annual Symposium Proceedings, pages 171.
- [2006 CROWLEY and MEDVEDEVA]: 2006; Rebecca S. CROWLEY, Olga MEDVEDEVA; An intelligent tutoring system for visual classification problem solving; Journal Artificial Intelligence in Medicine, volume 36, issue 1, pages 85-117, ISSN 0933-3657.
- [2010 CURILEM, VIZCARRA and POO]: 2010; G. H. CURILEM, B. VIZCARRA, A. M. POO, D. HUENTEMAN, D. HUENTEMAN, L. M. BRASIL; Design Methodology of an Intelligent Learning Environment Applied to the Non Violent Conflict Resolution Education; Journal Intelligent Tutoring Systems in E-Learning Environments: Design, Implementation and Evaluation: Design, Implementation and Evaluation, pages 27, ISSN 1616920092.
- [2006 D BAKER, CORBETT and KOEDINGER]: 2006; Ryan S. J. D BAKER, Albert T. CORBETT, Kenneth R. KOEDINGER, Shelley EVENSON, Ido ROLL, Angela Z. WAGNER, Meghan NAIM, Jay RASPAT; Adapting to when students game an intelligent tutoring system; International Conference on Intelligent Tutoring Systems, pages 392-401, Springer.
- [1996 DE BARROS COSTA and PERKUSISCH]: 1996; Evandro DE BARROS COSTA, Angelo PERKUSISCH; Modeling the cooperative interactions in a teaching/learning situation; International Conference on Intelligent Tutoring Systems, Springer Verlag, Heidelberg.
- [1991 DE JONG]: 1991; Ton DE JONG; Learning and instruction with computer simulations; Journal Education and Computing, volume 6, issue 3, pages 217-229, ISSN 0167-9287.
- [2006 DUBOIS, NKAMBOU and HOHMEYER]: 2006; Daniel DUBOIS, Roger NKAMBOU, Patrick HOHMEYER; How “Consciousness” Allows a Cognitive Tutoring Agent Make Good Diagnosis During Astronauts’ Training; International Conference on Intelligent Tutoring Systems, pages 154-163, Springer Verlag, Berlin, Heidelberg.
- [2017a DUDENVERLAG]: 2017a; DUDENVERLAG; Definition Wissen; Duden webpage http://www.duden.de/rechtschreibung/Wissen.
- [2017b DUDENVERLAG]: 2017b; DUDENVERLAG; Definition Wissenstransfer; Duden webpage http://www.duden.de/rechtschreibung/Wissenstransfer.
- [2017c DUDENVERLAG]: 2017c; DUDENVERLAG; Definition Lernen; Duden webpage http://www.duden.de/rechtschreibung/lernen.
- [2018a DUDENVERLAG]: 2018a; DUDENVERLAG; Definition Lerner; Duden webpage https://www.duden.de/rechtschreibung/Lerner.
- [2018b DUDENVERLAG]: 2018b; DUDENVERLAG; Definition Lernender; Duden webpage https://www.duden.de/rechtschreibung/Lernender.
- [2014 EAGLE and BARNES]: 2014; Michael EAGLE, Tiffany BARNES; Modeling student dropout in tutoring systems; Conference Intelligent Tutoring Systems, Publisher Springer, pages 676-678, ISBN 331907220X.

- [2006 EL-KECHAÏ and DESPRÉS]: 2006; Naïma EL-KECHAÏ, Christophe DESPRÉS; A plan recognition process, based on a task model, for detecting learner's erroneous actions; International Conference on Intelligent Tutoring Systems, pages 329-338, Springer.
- [1998 EL-SHEIKH and STICKLEN]: 1998; Eman EL-SHEIKH, Jon STICKLEN; A framework for developing intelligent tutoring systems incorporating reusability; International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Springer Berlin Heidelberg.
- [2018 FAKT SOFTWARE and DAEDALIC ENTERTAINMENT]: 2018; FAKT SOFTWARE, DAEDALIC ENTERTAINMENT; Crazy machines; http://www.crazy-machines.com.
- [2014 FENG, ROSCHELLE and HEFFERNAN]: 2014; Mingyu FENG, Jeremy ROSCHELLE, Neil HEFFERNAN, Janet FAIRMAN; Implementation of an intelligent tutoring system for online homework support in an efficacy trial; International Conference on Intelligent Tutoring Systems, pages 561-566, Springer, Cham.
- [1991 FERNALD and JORDAN]: 1991; Peter S. FERNALD, Elizabeth A. JORDAN; Programmed instruction versus standard text in introductory psychology; Teaching of Psychology 1.4, pages 205-2011.
- [2007 FERRARO]: 2007; Jack FERRARO; The strategic project leader: Mastering service-based project leadership; Book, Auerbach Publications.
- [2008 FORTIN, LEBEAU and ABDESSEMED]: 2008; Mikaël FORTIN, Jean-François LEBEAU, Amir ABDESSEMED, François COURTEMANCHE, André MAYERS; A Standard Method of Developing User Interfaces for a Generic ITS Framework; International Intelligent Tutoring Systems conference, pages 312-322 Springer Berlin Heidelberg.
- [2010 FOSSATI, EUGENIO and OHLSSON]: 2010; Davide FOSSATI, Barbara Di EUGENIO, Stellan OHLSSON, Christopher BROWN, Lin CHEN; Generating proactive feedback to help students stay on track; International Conference on Intelligent Tutoring Systems, pages 315-317, Springer Verlag, Berlin, Heidelberg.
- [2000a FREEDMAN, ALI and MCROY]: 2000a; Reva FREEDMAN, Syed S. ALI, Susan MCROY; What is an intelligent tutoring system?; Intelligence 11.3, pages 15-16.
- [2000b FREEDMAN, PENSTEIN and RINGENBERG]: 2000b; Reva FREEDMAN, Rosé PENSTEIN, Michael A. RINGENBERG, Kurt VAN LEHN; ITS tools for natural language dialogue: A domain-independent parser and planner; International Conference on Intelligent Tutoring Systems, pages 433-442, Springer Verlag, Berlin, Heidelberg.
- [2001 FUCHS-KITTOWSKI]: 2001; Klaus FUCHS-KITTOWSKI; Wissens-Ko-Produktion. Verarbeitung, Verteilung und Entstehung von Informationen in kreativ-lernenden Organisationen; Stufen zur Informationsgesellschaft für alle, Festschrift zum 65. Geburtstag.
- [2007 FUNDA and KADIR]: 2007; Dag FUNDA, Erkan KADIR; Realizing the Personalized Learning Paths in a LMS; Online Submission, Paper presented at the International Educational Technology (IETC) Conference 7th, Nicosia, Turkish Republic of Northern Cyprus.
- [1995 GACEK, ABD-ALLAH and BRADFORD]: 1995; Cristina GACEK, Ahmed ABD-ALLAH, Clark BRADFORD, Barry BOEHM; On the definition of software system architecture; In Proceedings of the First International Workshop on Architectures for Software Systems, pages 85-94.
- [2002 GAMBOA and FRED]: 2002; Hugo GAMBOA, Ana FRED; Designing intelligent tutoring systems: a bayesian approach; Enterprise Information Systems III, Springer Verlag New York, pages 146-152.
- [1994 GAMMA, JOHNSON and HELM]: 1994; Erich GAMMA, Ralph JOHNSON, Richard HELM, John VLISSIDES; Design Patterns: Elements of Reusable Object-Oriented Software; Addison-Wesley, ISBN 0-201-63361-2.

- [1993 GARLAN and SHAW]: 1993; David GARLAN, Mary SHAW; An introduction to software architecture; Advances in software engineering and knowledge engineering, pages 1-39.
- [1994 GARLAN and SHAW]: 1994; David GARLAN, Mary SHAW; An introduction to software architecture; Carnegie Mellon University, Software Engineering Institute, volume 1.
- [1995 GARLAN and PERRY]: 1995; David GARLAN, Dewayne E. PERRY; Introduction to the special issue on software architecture; IEEE Trans. Software Eng. 21, number 4, pages 269-274.
- [2008 GERRIG and ZIMBARDO]: 2008; Richard J. GERRIG, Philip G. ZIMBARDO; Psychologie; Book, Pearson Deutschland GmbH.
- [2000 GERTNER and VAN LEHN]: 2000; Abigail S. GERTNER, Kurt VAN LEHN; Andes: A coached problem solving environment for physics; International Conference on Intelligent Tutoring Systems, pages 133-142, Springer Verlag, Berlin, Heidelberg.
- [2000 GERTNER and VAN LEHN]: 2000; Abigail S. GERTNER, Kurt VAN LEHN; Andes: A Coached Problem Solving Environment for Physics; Intelligent Tutoring Systems: 5th International Conference, pages 131-142, Springer-Verlag, Berlin Heidelberg.
- [2010 GLUCK, MERCADO and MYERS]: 2010; Mark A. GLUCK, Erduardo MERCADO, Catherine E. MYERS; Lernen und Gedächtnis – Vom Gehirn zum Verhalten; Springer Verlag.
- [2014 GONZALEZ-SANCHEZ, CHAVEZ-ECHEAGARAY and VAN LEHN]: 2014; Javier GONZALEZ-SANCHEZ, Maria Elena CHAVEZ-ECHEAGARAY, Kurt VAN LEHN, Winslow BURLESON, Sylvie GIRARD, Yoalli HIDALGO-PONTET, Lishan ZHANG; A System Architecture for Affective Meta Intelligent Tutoring Systems; Conference Intelligent Tutoring Systems, Publisher Springer, pages 529-534, ISBN 331907220X.
- [2004 GOODKOVSKY]: 2004; Vladimir A. GOODKOVSKY; Intelligent Tutoring System US6807535 B2 or US20020107681 A1; US Patent and Trademark Office http://patft.uspto.gov/netacgi/nph-Parser?Sect2=PTO1&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&d=PALL&RefSrch=yes&Query=PN/6807535.
- [2020 GOOGLE]: 2020; GOOGLE; Google Assistant; https://assistant.google.com.
- [2005 GRAESSER, CHIPMAN and HAYNES]: 2005; Arthur C. GRAESSER, Patrick CHIPMAN, Brian C. HAYNES, Andrew OLNEY; AutoTutor: An Intelligent Tutoring System With Mixed-Initiative Dialogue; IEEE Transactions on education, volume 48, number 4.
- [2016 GRAF VON MALOTKY and MARTENS]: 2016; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; Framework for intelligent teaching and training systems - a study of systems; 13th International Conference Cognition and Exploratory Learning in the Digital Age (CELDA), ERIC number ED571423, ISBN 978-989-8533-55-5.
- [2017 GRAF VON MALOTKY, NAGORSNICK and NICOLAY]: 2017; Nikolaj Troels GRAF VON MALOTKY, Marian NAGORSNICK, Robin NICOLAY, Alke MARTENS; Bringing Hyperspectral Imaging into Clinical Practice with Computer Science Methods; Biomedizinische Technik / Biomedical Engineering Journal (BMT), ISSN 1862-278X, OCLC 163426070.
- [2017a GRAF VON MALOTKY, NAGORSNICK and MARTENS]: 2017a; Nikolaj Troels GRAF VON MALOTKY, Marian NAGORSNICK, Alke MARTENS; Hyperspectral Wound Diagnostic – Analysis of Clinical Practice and Diagnosis Support; 62. Jahrestagung Deutsche Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e. V. (GMDS), publisher gms.
- [2017a GRAF VON MALOTKY, NICOLAY and MARTENS]: 2017a; Nikolaj Troels GRAF VON MALOTKY, Robin NICOLAY, Alke MARTENS; Synthesis of pedagogical annotations; 9th annual International Conference on Education and New Learning Technologies

(EDULEARN), EDULEARN17 Proceedings, ISBN 978-84-697-3777-4, ISSN 2340-1117, DOI 10.21125/edulearn.2017, pages 3655-3661.

- [2017b GRAF VON MALOTKY, NAGORSNICK and MARTENS]: 2017b; Nikolaj Troels GRAF VON MALOTKY, Marian NAGORSNICK, Alke MARTENS; Hyperspectral Imaging and Clinical Practice – Approach to Integrate a New Diagnostic Too; 51. Jahrestagung der BIOMEDIZINISCHEN TECHNIK und Dreiländertagung der MEDIZINISCHEN PHYSIK (BMTMedPhys), Deutsche Gesellschaft für Biomedizinische Technik.
- [2017b GRAF VON MALOTKY, NICOLAY and MARTENS]: 2017b; Nikolaj Troels GRAF VON MALOTKY, Robin NICOLAY, Alke MARTENS; Centralizing the Teaching Process in Intelligent Tutoring System Architectures; 19th International Conference on Advanced Learning Technologies (ICALT), Open Science Index, Educational and Pedagogical Sciences, volume 11, number 10, ISNI 0000000091950263.
- [2019a GRAF VON MALOTKY and MARTENS]: 2019a; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; A comparable foundation for ITS research loop and subfields; 11th annual International Conference on Education and New Learning Technologies (EDULEARN), pages 1402-1409, DOI 10.21125/edulearn.2019.0425.
- [2019b GRAF VON MALOTKY and MARTENS]: 2019b; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; Analyzing the usage of the classical ITS software architecture and refining it; 15th international conference on intelligent tutoring systems (ITS2019), pages 40-46, DOI 10.1007/978-3-030-22244-4_6, Springer, Cham, Online ISBN 978-3-030-22244-4, Print ISBN 978-3-030-22243-7.
- [2020a GRAF VON MALOTKY and MARTENS]: 2020a; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; General ITS software architecture and framework; 16th international conference on intelligent tutoring systems (ITS2020), Springer.
- [2020b GRAF VON MALOTKY and MARTENS]: 2020b; Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; DigiCare - Ein intelligentes Lehr-/Lernsystem für Gesundheitsberufe; 18. Fachtagung Bildungstechnologien (DELFI), Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2020.
- [2009 GUNEL and ASLIYAN]: 2009; Korhan GUNEL, Rifat ASLIYAN; Determining Difficulty of Questions in Intelligent Tutoring Systems; Turkish Online Journal of Educational Technology (TOJET), volume 8, number 3, pages 14-21.
- [2010 GUTIERREZ-SANTOS, MAVRIKIS and MAGOULAS]: 2010; Sergio GUTIERREZ-SANTOS, Manolis MAVRIKIS, George MAGOULAS; Layered development and evaluation for intelligent support in exploratory environments: the case of microworlds; Conference International Conference on Intelligent Tutoring Systems, Publisher Springer, pages 105-114.
- [2006 HÄMÄLÄINEN and VINNI]: 2006; Wilhelmiina HÄMÄLÄINEN, Mikko VINNI; Comparison of machine learning methods for intelligent tutoring systems; International Conference on Intelligent Tutoring Systems, pages 525-534. Springer.
- [2005 HAN, LEE and JO]: 2005; Sun-Gwan HAN, Soon-Geun LEE, Geun-Sik JO; Case-based tutoring systems for procedural problem solving on the www; Journal Expert Systems with applications 29, number 3, pages 573-582.
- [2000 HARRER]: 2000; Andreas Georg HARRER; Unterstützung von Lerngemeinschaften in verteilten intelligenten Lehrsystemen; PhD diss., Technische Universität München.
- [2003 HARRER]: 2003; Andreas HARRER; Software engineering methods for re-use of components and design in educational systems; International Journal of Computers and Applications 25.

- [2006 HARRER and MARTENS]: 2006; Andreas HARRER, Alke MARTENS; Towards a Pattern Language for Intelligent Teaching and Training Systems; Springer Berlin / Heidelberg.
- [1973 HARTLEY and SLEEMAN]: 1973; J. R. HARTLEY, Derek H. SLEEMAN; Towards more intelligent teaching systems; International Jurnal of Man-Machine Studies 5.2, pages 215-236, DOI 10.1016/S0020-7373(73)80033-1.
- [2004 HATZILYGEROUDIS and PRENTZAS]: 2004; Ioannis HATZILYGEROUDIS, Jim PRENTZAS; Using a hybrid rule-based approach in developing an intelligent tutoring system with knowledge acquisition and update capabilities; Expert systems with applications, volume 26, issue 4, pages 477-492, ISSN 0957-4174.
- [1995 HAYES-ROTH, PFLEGER and LALANDA]: 1995; Barbara HAYES-ROTH, Karl PFLEGER, Philippe LALANDA, Philippe MORIGNOT, Marko BALABANOVIC; A domain-specific software architecture for adaptive intelligent systems; IEEE Transactions on software engineering 21, number 4, pages 288-301.
- [1998 HEFFERNAN]: 1998; Neil T. HEFFERNAN; Intelligent tutoring systems have forgotten the tutor: Adding a cognitive model of human tutors; CHI 98 Conference Summary on Human Factors in Computing Systems, ACM, pages 50-51.
- [2002 HEFFERNAN and KOEDINGER]: 2002; Neil T. HEFFERNAN, Kenneth R. KOEDINGER; An intelligent tutoring system incorporating a model of an experienced human tutor; n International Conference on Intelligent Tutoring Systems, pages 596-608, Springer Verlag, Berlin, Heidelberg, 2002.
- [2012 HERCULANO-HOUZEL]: 2012; Suzana HERCULANO-HOUZEL; The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost; Journal Proceedings of the National Academy of Sciences, volume 109, issue Supplement 1, pages 10661-10668, ISSN 0027-8424.
- [2000 HILLIARD]: 2000; Rich HILLIARD; Recommended practice for architectural description of software-intensive systems; IEEE, http://standards.ieee.org, number 12, pages 16-20, ieee-std-1471-2000.
- [2010 HIRASHIMA, YAMAMOTO and WAKI]: 2010; Tsukasa HIRASHIMA, Sho YAMAMOTO, Hiromi WAKI; An interactive learning environment for problem-changing exercise; International Conference on Intelligent Tutoring Systems. Springer, Berlin, Heidelberg.
- [2002 HODGINS and DUVAL]: 2002; Wayne HODGINS, Erik DUVAL; Draft standard for learning technology-Learning Object Metadata-ISO/IEC 11404; IEEE P1484. 12.2/D1, https://www.ieeeltsc.org/wg20Comp/wg20rcdfolder/IEEE_1484.20.1.D3.pdf.
- [1997 HOFMANN]: 1997; Thomas HOFMANN; Interaktives Lernen mit dem Internet: Theoretische Grundlagen und praktische Entwicklung von internetbasierten Lernumgebungen; Diplomarbeit at Georg-Simon-Ohm-Fachhochschule Nürnberg http://paedpsych.jku.at:4711/LEHRTEXTE/Internetlernen/index.html / https://perma.cc/CS4A-JCJX.
- [2003 IEEE]: 2003; IEEE; 1484.1-2003 - IEEE Standard for Learning Technology - Learning Technology Systems Architecture (LTSA); Learning Technology Standards Committee of the IEEE Computer Society.
- [2018 IEEE STANDARDS ASSOCIATION]: 2018; IEEE STANDARDS ASSOCIATION; ANSI/IEEE 1484.1-2003 - IEEE Standard for Learning Technology - Learning Technology Systems Architecture (LTSA); http://standards.ieee.org/findstds/standard/1484.1-2003.html.
- [1994 IKEDA and MIZOGUCHI]: 1994; Mitsuru IKEDA, Riichiro MIZOGUCHI; FITS: a framework for ITS - a computational model of tutoring; Journal of Interactive Learning Research 5, number 3, page 319.

- [2014 IMRAN]: 2014; Hazra IMRAN; A Rule-Based Recommender System to Suggest Learning Tasks.; Conference Intelligent Tutoring Systems, Springer International Publishing.
- [2018 IN2013DOLLARS.COM]: 2018; IN2013DOLLARS.COM; Inflation Calculator; http://www.in2013dollars.com.
- [2018 INSPIRATION SOFTWARE INC]: 2018; INSPIRATION SOFTWARE INC; Inspiration; http://www.inspiration.com/Inspiration.
- [1995 JACKSON and BOASSON]: 1995; Ken JACKSON, Maarten BOASSON; The benefits of good architectural style in the engineering of computer based systems; Systems Engineering of Computer Based Systems, 1995., Proceedings of the 1995 International Symposium and Workshop, IEEE, pages 103-113.
- [2004 JEREMIC, DEVEDZIC and GASEVIC]: 2004; Z. JEREMIC, Vladan DEVEDZIC, D. GASEVIC; An intelligent tutoring system for learning design patterns; International Workshop on Adaptative Hypermedia and Collaborative Web-based Systems (AHCW).
- [2001 JERMANN, SOLLER and MUEHLENBROCK]: 2001; Patrick JERMANN, Amy SOLLER, Martin MUEHLENBROCK; From mirroring to guiding: A review of the state of the art technology for supporting collaborative learning; European Conference on Computer-Supported Collaborative Learning EuroCSCL-2001, Maastricht Netherlands, pages 324-331, HAL hal-00197377.
- [2009 JOHNSON, PHILLIPS and CHASE]: 2009; Benny G. JOHNSON, Fred PHILLIPS, Linda G. CHASE; An intelligent tutoring system for the accounting cycle: Enhancing textbook homework with artificial intelligence; Journal of Accounting Education 27, number 1, pages 30-39.
- [2012 JRAIDI and CHALFOUN]: 2012; Imène JRAIDI, Pierre CHALFOUN; Implicit strategies for intelligent tutoring systems; Conference Intelligent Tutoring Systems, Publisher Springer, pages 1-10, ISBN 3642309496.
- [2005 KAFKAS, BAYRAM and YARATAN]: 2005; S. KAFKAS, Zeki BAYRAM, Hüseyin YARATAN; A User Friendly Intelligent Algebra Tutor; Proceedings of the 5th International Educational Technologies Conference (IETC).
- [2005 KARAMPIPERIS and SAMPSON]: 2005; Pythagoras KARAMPIPERIS, Demetrios SAMPSON; Adaptive learning resources sequencing in educational hypermedia systems; Journal of Educational Technology & Society, volume 8, issue 4, pages 128-147, ISSN 1176-3647.
- [2016 KATADA]: 2016; Fusa KATADA; Communication Vulnerability in the Digital Age: A Missed Concern in Constructivism; International Association for Development of the Information Society.
- [1987 KEARSLEY]: 1987; Greg P. KEARSLEY; Artificial Intelligence and instruction: Applications and methods; Addison-Wesley Longman Publishing Co..
- [1993 KEARSLEY]: 1993; Greg P. KEARSLEY; Intelligent agents and instructional systems: Implications of a new paradigm; Journal of Interactive Learning Research 4, Number 4, page 295.
- [2006 KELLY and TANGNEY]: 2006; Declan KELLY, Brendan TANGNEY; Using multiple intelligence informed resources in an adaptive system; International Conference on Intelligent Tutoring Systems, Publisher Springer, pages 412-421.
- [2002 KERRES]: 2002; Michael KERRES; Online-und Präsenzelemente in hybriden Lernarrangements kombinieren; Handbuch E-Learning, Fachverlag Deutscher Wirtschaftsdienst, Köln.

- [2017 KNOLL]: 2017; Michael KNOLL; „Learning by doing“. Zur Genese eines pädagogischen Slogans; Mythen–Irrtümer–Unwahrheiten. Essays über das „Valsche “in der Pädagogik. Hrsg. Grunder, Hans-Ulrich. Bad Heilbrunn, pages 127-132.
- [2007 KNUTH]: 2007; Donald E. KNUTH; Computer programming as an art; ACM Turing award lectures, page 1974.
- [2005 KODAGANALLUR, WEITZ and ROSENTHAL]: 2005; Viswanathan KODAGANALLUR, Rob R. WEITZ, David ROSENTHAL; A comparison of model-tracing and constraint-based intelligent tutoring paradigms; International Journal of Artificial Intelligence in Education 15, number 2 (2005): 117-144.
- [2010 KODAVALI, GILBERT and BLESSING]: 2010; Sateesh Kumar KODAVALI, Stephen GILBERT, Stephen B. BLESSING; Expansion of the xPST framework to enable non-programmers to create intelligent tutoring systems in 3D game environments; International Conference on Intelligent Tutoring Systems, pages 365-367. Springer, Berlin, Heidelberg.
- [1997 KOEDINGER, ANDERSON and HADLEY]: 1997; Kenneth R. KOEDINGER, John R. ANDERSON, William H. HADLEY, Mary A. MARK; Intelligent Tutoring Goes To School in the Big City; International Journal of Artificial Intelligence in Education 8, pages 30-43..
- [2009 KOEDINGER, ALEVEN and MCLAREN]: 2009; Kenneth R. KOEDINGER, Vincent ALEVEN, Bruce M. MCLAREN, Jonathan SEWALL; Example-Tracing Tutors: A New Paradigm for Intelligent Tutoring Systems; Authoring Intelligent Tutoring Systems.
- [1994 KOGUT and CLEMENTS]: 1994; Paul KOGUT, Paul CLEMENTS; The software architecture renaissance; Crosstalk-The Journal of Defense Software Engineering 7, number 11, pages 1-5.
- [2006 KOJIMA and MIWA]: 2006; Kazuaki KOJIMA, Kazuhisa MIWA; Evaluation of a system that generates word problems through interactions with a user; International Conference on Intelligent Tutoring Systems, pages 124-133, Springer Verlag, Berlin, Heidelberg.
- [2018 KOKKINAKIS]: 2018; Sofie Johansson KOKKINAKIS; IKITS - An Intelligent Tutoring System for Chinese; https://spraakbanken.gu.se/personal/sofie/IKITS/IKITSeng.html.
- [2009 KÖNIG and BLÖMEKE]: 2009; Johannes KÖNIG, Sigrid BLÖMEKE; Pedagogic knowledge of future teachers; Journal Zeitschrift für Erziehungswissenschaft, volume 12, issue 3, pages 499-527, ISSN 1862-5215, DOI 10.1007/s11618-009-0085-z.
- [2007 KRAVCIK and GASEVIC]: 2007; Milos KRAVCIK, Dragan GASEVIC; Leveraging the Semantic Web for Adaptive Education; Journal of Interactive Media in Education.
- [1994 KRUCHTEN and THOMPSON]: 1994; Phillipe KRUCHTEN, Christopher J. THOMPSON; An object-oriented, distributed architecture for large-scale Ada systems; Proceedings of the conference on TRI-Ada'94, ACM, pages 262-271.
- [2016 KULIK and FLETCHER]: 2016; James A. KULIK, J. D. FLETCHER; Effectiveness of Intelligent Tutoring Systems: A Meta-Analytic Review ; Review of Educational Research 86, number 1, pages 42-78..
- [2018 LAMINAR RESEARCH]: 2018; LAMINAR RESEARCH; X Plane; http://www.x-plane.com.
- [1990 LANE]: 1990; Thomas G. LANE; Studying software architecture through design spaces and rules; Final Report.
- [1995 LAWSON, KIROVA and ROSSAK]: 1995; H. F. LAWSON, Vassilka KIROVA, William ROSSAK; A refinement of the ecbs architecture constituent; Systems Engineering of Computer Based Systems, 1995., Proceedings of the 1995 International Symposium and Workshop, IEEE, pages 95-102.

- [2006 LEE, CHO and CHOI]: 2006; Youngseok LEE, Jungwon CHO, Byung-Uk CHOI; An intelligent tutoring system based on a multi-modal environment for the 300-certification program of english conversation; International Conference on Intelligent Tutoring Systems, pages 778-780, Springer Verlag, Berlin, Heidelberg.
- [1998 LELOUCHE]: 1998; Ruddy LELOUCHE; The successive contributions of computers to education: A survey; European Journal of Engineering Education, volume 23, issue 3, pages 297-308, ISSN 0304-3797.
- [1999 LELOUCHE]: 1999; Ruddy LELOUCHE; Intelligent tutoring systems from birth to now; Journal KI, volume 13, number 4, pages 5-11.
- [2003 LELOUCHE and LY]: 2003; Ruddy LELOUCHE, T. T. LY; Using a framework in the development of an intelligent tutoring system; Information Reuse and Integration, IRI 2003. IEEE International Conference, pages 291 - 298, DOI 10.1109/IRI.2003.1251428.
- [2012 LIN, WANG and CHAO]: 2012; Hao-Chiang Koong LIN, Cheng-Hung WANG, Ching-Ju CHAO, Ming-Kuan CHIEN; Employing Textual and Facial Emotion Recognition to Design an Affective Tutoring System; Turkish Online Journal of Educational Technology - TOJET, v11 n4 p418-426 Oct 2012.
- [2004 LITMAN and SILLIMAN]: 2004; Diane J. LITMAN, Scott SILLIMAN; ITSPOKE: an intelligent tutoring spoken dialogue system; Conference Paper, Demonstration Papers at HLT-NAACL, Association for Computational Linguistics, Boston, Massachusetts, pages 5-8.
- [2006 LITMAN, ROSÉ and FORBES-RILEY]: 2006; Diane J. LITMAN, Carolyn P. ROSÉ, Kate FORBES-RILEY, Kurt VAN LEHN, Dumisizwe BHEMBE, Scott SILLIMAN; Spoken versus typed human and computer dialogue tutoring; International Journal of Artificial Intelligence in Education, number 16, pages 145-170.
- [2004 LOCKEE, MOORE and BURTON]: 2004; Barbara LOCKEE, D. MOORE, John BURTON; Foundations of programmed instruction; Handbook of research on educational communications and technology, pages 545-569.
- [2000 LOS ARCOS, MULLER and FUENTE]: 2000; José Luis LOS ARCOS, Wolfgang MULLER, Oscar FUENTE, Leire ORÚE, Eder ARROYO, Igor LEAZNIBARRUTIA, Judit SANTANDER; Lahystotrain: Integration of virtual environments and its for surgery training; International Conference on Intelligent Tutoring Systems, pages 43-52, Springer Verlag, Berlin, Heidelberg.
- [2001 LUNDGREN-CAYROL, PAQUETTE and MIARA]: 2001; Karin LUNDGREN-CAYROL, Gilbert PAQUETTE, Alexis MIARA, Frederick BERGERON, Frederick BERGERON, Jacques RIVARD, Ioan ROSCA; Explor@ Advisory Agent: Tracing the Student's Trail.; In WebNet World Conference on the WWW and Internet Proceedings Orlando, FL see IR 021 310.
- [1992 LUSTI]: 1992; Markus LUSTI; Intelligente tutorielle Systeme; Publisher Oldenbourg, ISBN 3486216708.
- [2011 MACIUSZEK and MARTENS]: 2011; Dennis MACIUSZEK, Alke MARTENS; A Reference Architecture for Game-based Intelligent Tutoring; Handbook of Research on Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches, IGI Global, ISBN 978-1609604950.
- [2005 MAIR]: 2005; Daniela MAIR; E-Learning - Das Drehbuch: Handbuch für Medienautoren und Projektleiter; Book, Springer Verlag.
- [2017 MAKRIDAKIS]: 2017; Spyros MAKRIDAKIS; The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms; Futures 90, pages 46-60.
- [2011 MANOUSELIS, DRACHSLER and VUORIKARI]: 2011; NIKOS MANOUSELIS, HENDRIK DRACHSLER, RIINA VUORIKARI, Hans HUMMEL, Rob KOPER; Recommen-

der Systems in Technology Enhanced Learning; Recommender Systems Handbook, Springer US, ISBN 978-0-387-85819-7.

- [2015 MARK]: 2015; Richards MARK; Software architecture patterns; O'Reilly Media Incorporated.
- [2003 MARTENS]: 2003; Alke MARTENS; Centralize the tutoring process in intelligent tutoring systems; International conference proceedings of the 5th New Educational Environments ICNEE, Lucerne, Switzerland.
- [2004 MARTENS]: 2004; Alke MARTENS; Ein Tutoring Prozess Modell für fallbasierte Intelligente Tutoring Systeme; Dissertationen zur Künstlichen Intelligenz Band 281, ISBN 3-89838-281-8.
- [2008 MARTENS, DIENER and MALO]: 2008; Alke MARTENS, Holger DIENER, Steffen MALO; Game-based Learning with Computers - Learning, Simulations, and Games; Journal Transactions on Edutainment, pages 172-190.
- [2010 MARTENS and HARRER]: 2010; Alke MARTENS, Andreas HARRER; Intelligent Tutoring System Architecture Rebuilt – A pattern approach; Intelligent Tutoring Systems in eLearning Environments, Information Science Reference, US, pages 70–86, chapter 4, ISBN 978-1-61692-008-1.
- [2015 MARTENS]: 2015; Alke MARTENS; Software Engineering and Modeling in TEL; The New Development of Technology Enhanced Learning: Concept, Research and Best Practices, Lecture Notes in Educational Technology, Springer Verlag, Berlin, Heidelberg, Germany.
- [2017a MARTENS, NICOLAY and GRAF VON MALOTKY]: 2017a; Alke MARTENS, Robin NICOLAY, Nikolaj Troels GRAF VON MALOTKY; Integration neuer diagnostischer Verfahren in den klinischen Alltag; Symposium Hyperspektrale Bildgebung in der Medizin (HSI), https://tu-dresden.de/ing/elektrotechnik/ibmt/ressourcen/dateien/ibmt/tagungen/tagungen-2017/hsi-symposium/2017_HSI_Symposium.pdf.
- [2017b MARTENS, NICOLAY and GRAF VON MALOTKY]: 2017b; Alke MARTENS, Robin NICOLAY, Nikolaj Troels GRAF VON MALOTKY; Hyperspectral imaging and clinical practice – approach to integrate a new diagnostic tool; forschungscamp, Universität Rostock.
- [2002 MARTIN]: 2002; Robert C. MARTIN; Agile software development: principles, patterns, and practices; Book, Prentice Hall, Pearson Education Incorporated.
- [2010 MATSUDA, KEISER and RAIZADA]: 2010; Noboru MATSUDA, Victoria KEISER, Rohan RAIZADA, Gabriel STYLIANIDES, William W. COHEN, Kenneth R. KOEDINGER; Learning by Teaching SimStudent; Intelligent Tutoring Systems, page 449.
- [2015 MATSUDA, COHEN and KOEDINGER]: 2015; Noboru MATSUDA, William W. COHEN, Kenneth R. KOEDINGER; Teaching the teacher: tutoring SimStudent leads to more effective cognitive tutor authoring; International Journal of Artificial Intelligence in Education 25, number 1, pages 1-34..
- [2000 MATSUURA, OGATA and YANO]: 2000; Kenji MATSUURA, Hiroaki OGATA, Yoneo YANO; Agent's contribution for an Asynchronous Virtual Classroom; International Conference on Intelligent Tutoring Systems, pages 344-353, Springer Verlag, Berlin, Heidelberg.
- [2000a MAYO and MITROVIC]: 2000a; Michael MAYO, Antonija MITROVIC; Using a probabilistic student model to control problem difficulty; International Conference on Intelligent Tutoring Systems, pages 262-271, Springer Verlag, Berlin, Heidelberg.
- [2000b MAYO, MITROVIC and MCKENZIE]: 2000b; Michael MAYO, Antonija MITROVIC, Jane MCKENZIE; CAPIT: An intelligent tutoring system for capitalisation and punctuati-

on; IEEE, Advanced Learning Technologies, IWALT Proceedings, International Workshop, pages 151-154.

- [2006 MCLAREN, LIM and GAGNON]: 2006; Bruce M. MCLAREN, Sung-Joo LIM, France GAGNON, Davin YARON, Kenneth R. KOEDINGER; Studying the effects of personalized language and worked examples in the context of a web-based intelligent tutor; International Conference on Intelligent Tutoring Systems, pages 318-328, Springer, Berlin, Heidelberg.
- [2010 MEDVIDOVIC and TAYLOR]: 2010; Nenad MEDVIDOVIC, Richard N. TAYLOR; Software architecture: foundations, theory, and practice; Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, ACM, pages 471-472.
- [2004 MELIS and SIEKMANN]: 2004; Erica MELIS, Jörg SIEKMANN; Activemath: An intelligent tutoring system for mathematics; Artificial Intelligence and Soft Computing (ICAISC) , Springer, pages 91-101.
- [2009 MENDICINO, RAZZAQ and HEFFERNAN]: 2009; Michael MENDICINO, Leena RAZZAQ, Neil T. HEFFERNAN; Comparison of Traditional Homework with Computer Supported Homework; Journal of Research on Technology in Education 41(3), pages 331–359.
- [2000 MICHAUD, MCCOY and PENNINGTON]: 2000; Lisa N. MICHAUD, Kathleen F. MCCOY, Christopher A. PENNINGTON; An intelligent tutoring system for deaf learners of written English; Proceedings of the fourth international ACM conference on Assistive technologies, pages 92-100.
- [2019 MICROSOFT CORPORATION]: 2019; MICROSOFT CORPORATION; Design Patterns: Model View Presenter; https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/august/design-patterns-model-view-presenter.
- [2020 MICROSOFT CORPORATION]: 2020; MICROSOFT CORPORATION; Cortana; https://www.microsoft.com/de-de/windows/cortana.
- [1998 MITROVIC]: 1998; Antonija MITROVIC; Learning SQL with a computerized tutor; In ACM SIGCSE Bulletin, vol. 30, number 1, pages 307-311.
- [2000 MITROVIC and SURAWEERA]: 2000; Antonija MITROVIC, Pramuditha SURAWEERA; Evaluating an animated pedagogical agent; International Conference on Intelligent Tutoring Systems, pages 73-82, Springer Verlag, Berlin, Heidelberg.
- [2009 MITROVIC, BRENT and SURAWEERA]: 2009; Antonija MITROVIC, Martin BRENT, Pramuditha SURAWEERA, Nancy MILIK, Jay HOLLAND, Nicholas MCGUIGAN; ASPIRE: an authoring system and deployment environment for constraint-based tutors; International Conference on Artificial Intelligence in Education, pages 155-188, number 2, Springer, Berlin, Heidelberg.
- [2012 MITROVIC and MATHEWS]: 2012; Antonija MITROVIC, Moffat MATHEWS; Exploring two strategies for teaching procedures; Conference Intelligent Tutoring Systems, Publisher Springer, pages 499-504, ISBN 3642309496.
- [2012 MITROVIC]: 2012; Antonija MITROVIC; Fifteen years of constraint-based tutors: what we have achieved and where we are going ; User modeling and user-adapted interaction 22, number 1-2, pages 39-72.
- [2019 MITROVIC and HOLLAND]: 2019; Antonija MITROVIC, J. HOLLAND; INVESTIGATING THE EFFECT OF VOLUNTARY USE OF AN INTELLIGENT TUTORING SYSTEM ON STUDENTS'LEARNING; 11th annual International Conference on Education and New Learning Technologies (EDULEARN), pages.
- [2009 MNKANDLA]: 2009; Ernest MNKANDLA; About Software Engineering Frameworks and Methodologies; AFRICON, pages 1-5, IEEE.

- [2002 MOEBUS, ALBERS and HARTMANN]: 2002; Claus MOEBUS, Bernd ALBERS, Stefan HARTMANN, Hein-Juergen THOLE, Jochen ZURBORG; Towards a Specification of Distributed and Intelligent Web Based Training Systems; Intelligent Tutoring Systems, Springer Berlin/Heidelberg.
- [1994 MORICONI and QIAN]: 1994; Mark MORICONI, Xiaolei QIAN; Correctness and composition of software architectures; ACM SIGSOFT Software Engineering Notes, vol. 19, number 5, pages 164-174.
- [1996 MURRAY]: 1996; Tom MURRAY; From story boards to knowledge bases: the first paradigm shift in making CAI intelligent.; Journal Informatik-Spektrum, volume 33, issue 2, pages 201-207, ISSN 0170-6012.
- [1999 MURRAY]: 1999; Tom MURRAY; Authoring Intelligent Tutoring Systems: An analysis of the state of the art; International Journal of Artificial Intelligence in Education (IJAIED) 10, pages 98-129, HAL hal-00197339.
- [2003 MURRAY]: 2003; Tom MURRAY; An Overview of Intelligent Tutoring System Authoring Tools: Updated analysis of the state of the art; Publisher Springer, Book Title Authoring tools for advanced technology learning environments, pages 491-544, ISBN/ISSN 9048164990.
- [2002 NEDIC, NEDIC and MACHOTKA]: 2002; Zorica NEDIC, Vladimir NEDIC, Jan MACHOTKA; Intelligent Tutoring System for teaching 1st year engineering; World Transactions on Engineering and Technology Education, volume 1, number 2, pages 241-243.
- [2008 NEJI, AMMAR and ALIMI]: 2008; Mahmoud NEJI, Mohamed Ben AMMAR, Adel M. ALIMI, Guy GOUARDÈRES; Agent-based framework for affective intelligent tutoring systems; International Conference on Intelligent Tutoring Systems, pages 665-667, Springer Verlag, Berlin, Heidelberg.
- [2018 NEXT IS GREAT SP Z O.O.]: 2018; NEXT IS GREAT SP Z O.O.; RobotSchool; http://www.robotschoolapp.com.
- [2014 NICOLAY]: 2014; Robin NICOLAY; Semantic Enhancement of Lecture Material; arXiv preprint arXiv:1412.2901.
- [2017 NICOLAY, GRAF VON MALOTKY and AUGE]: 2017; Robin NICOLAY, Nikolaj Troels GRAF VON MALOTKY, Tanja AUGE, Alke MARTENS; Autonomous semantic structuring of lecture topics Synthesis of knowledge models; 9th International Conference on Computer Supported Education CSEDU, volume 2, pages 349-355, SCITEPRESS, DOI 10.5220/0006367903490355.
- [2008 NIELSEN, WARD and MARTIN]: 2008; Rodney D. NIELSEN, Wayne WARD, James H. MARTIN; Automatic Generation of Fine-Grained Representations of Learner Response Semantics; International Conference on Intelligent Tutoring Systems, Springer Verlag, Heidelberg.
- [2010 NKAMBOU, MIZOGUCHI and BOURDEAU]: 2010; Roger NKAMBOU, Riichiro MIZOGUCHI, Jacqueline BOURDEAU; Advances in intelligent tutoring systems; Springer Science & Business Media, volume 308, ISBN 3642143628.
- [1990 NWANA]: 1990; Hyacinth S. NWANA; Intelligent tutoring systems: an overview; Journal Artificial Intelligence Review, volume 4, issue 4, pages 251-277, ISSN 0269-2821.
- [1983 O'SHEA and SELF]: 1983; Tim O'SHEA, John SELF; Learning & Teaching with Computers: Artificial Intelligence in Education; Artificial Intelligence in Education, Harvester Press Limited, ISBN 0135277701 & 9780135277706.
- [1984 O'SHEA, BORNAT and DU BOULAY]: 1984; Tim O'SHEA, Richard BORNAT, Benedict DU BOULAY, Marc EISENSTADT, Ian PAGE; Tools for creating intelligent computer

tutors; Proceedings of the international NATO symposium on Artificial and human intelligence, pages 181-199, Elsevier North-Holland Inc..

- [2018 OBJECT MANAGEMENT GROUP (OMG)]: 2018; OBJECT MANAGEMENT GROUP (OMG); ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5; https://www.omg.org/spec/UML/2.5/About-UML/.
- [2006 OERTEL]: 2006; Marcus OERTEL; Entwicklung eines Frameworks für Intelligente Lehr-/Lernsysteme; Diplomarbeit - Fakultät für Informatik und Elektrotechnik - Universität Rostock.
- [2006 OGAN, ALEVEN and JONES]: 2006; Amy OGAN, Vincent ALEVEN, Christopher JONES; Culture in the classroom: Challenges for assessment in ill-defined domains; Intelligent Tutoring Systems for Ill-Defined Domains, page 92.
- [1992 OHLSSON]: 1992; S. OHLSSON; Learning from performance errors; Psychological review 103, number 2, page 241.
- [2012 OLNEY, D'MELLO and PERSON]: 2012; Andrew M. OLNEY, Sidney D'MELLO, Natalie PERSON, Whitney CADE, Pattrick HAYS, Claire WILLIAMS, Blair LEHMANN, Arthur GRAESSER; Guru: a computer tutor that models expert human tutors; Conference Intelligent Tutoring Systems, Publisher Springer, pages 256-261, ISBN 3642309496.
- [2015 OLUWATOBI and OLURINOLA]: 2015; Stephen OLUWATOBI, Oluranti Isaiah OLURINOLA; Mobile learning in Africa: strategy for educating the poor; International Conference on African Developement Issues (CU-ICADI): Information and Communication Technology Track.
- [2014 PALOMINO, SILVEIRA and NAKAYAMA]: 2014; Cecilia Estela Giuffra PALOMINO, Ricardo Azambuja SILVEIRA, Marina Keiko NAKAYAMA; An Intelligent LMS Model Based on Intelligent Tutoring Systems; Conference Intelligent Tutoring Systems, Publisher Springer, pages 567-574, ISBN331907220X.
- [1980 PAPERT]: 1980; Seymour PAPERT; Mindstorms: Children, computers, and powerful ideas; Book, Basic Books Inc., 244 pages, ISBN 978-0-465-04627-0.
- [2010 PAQUETTE, LEBEAU and MAYERS]: 2010; Luc PAQUETTE, Jean-François LEBEAU, André MAYERS; Authoring Problem-Solving Tutors: A Comparison between ASTUS and CTAT; Springer Verlag, pages 377-405.
- [2010 PAVLIK and TOTH]: 2010; Philip PAVLIK, Joe TOTH; How to build bridges between intelligent tutoring system subfields of research; International Conference on Intelligent Tutoring Systems, pages 103-112, Springer, Berlin, Heidelberg.
- [1992 PERRY and WOLF]: 1992; Dewayne E. PERRY, Alexander L. WOLF; Foundations for the study of software architecture; ACM SIGSOFT Software engineering notes 17, number 4, pages 40-52.
- [2002 PERSON, GRAESSER and TUTORING RESEARCH GROUP]: 2002; Natalie PERSON, Arthur C. GRAESSER, TUTORING RESEARCH GROUP; Human or computer? AutoTutor in a bystander Turing test; International Conference on Intelligent Tutoring, pages 821-830, Springer, Berlin, Heidelberg.
- [2004 PETRINA]: 2004; Stephen PETRINA; Sidney Pressey and the automation of education, 1924-1934; Journal Technology and Culture, issue 2, volume 45, pages 305-330, ISSN 1097-3729.
- [2000 PEYLO]: 2000; Christoph PEYLO; Wissen- und Wissensvermittlung im Kontext von internetbasierten intelligenten Lehr- und Lernumgebungen; Aka.
- [1967 PHILCO-FORD CORPORATION]: 1967; PHILCO-FORD CORPORATION; The Year 1999 A.D. - A Prediction of the Future from 1967; Short film, http://www.imdb.com/title/

tt6294456/ and https://www.youtube.com/watch?feature=share&v=HI0to23KbKs&app=desktop.

- [2008 PINKWART, LYNCH and ASHLEY]: 2008; Niels PINKWART, Collin LYNCH, Kevin ASHLEY, Vincent ALEVEN; Re-evaluating LARGO in the classroom: Are diagrams better than text for teaching argumentation skills?; In International Conference on Intelligent Tutoring Systems, pages 90-100, Springer, Berlin, Heidelberg.
- [2013 PIROLLI and KAIRAM]: 2013; Peter PIROLLI, Sanjay KAIRAM; A knowledge-tracing model of learning from social tagging system; User Model User-Adap Inter 23 (2-3), pages 139-168, DOI 10.1007/s11257-012-9132-1.
- [2005 PRENSKY and BOWERS]: 2005; Marc PRENSKY, Jan-Cannon BOWERS; Serious games debate; Serious Games Summit.
- [2002 PRENTZAS, HATZILYGEROUDIS and GAROFALAKIS]: 2002; Jim PRENTZAS, Ioannis HATZILYGEROUDIS, John GAROFALAKIS; A web-based intelligent tutoring system using hybrid rules as its representational basis; Intelligent Tutoring Systems, Springer Berlin Heidelberg.
- [1963 PRESSEY]: 1963; Sidney L. PRESSEY; Teaching machine (and learning theory) crisis; Journal of Applied Psychology 47, number 1, DOI 10.1037/h0047740.
- [1992 PUPPE]: 1992; Frank PUPPE; Intelligente Tutoringsysteme; Informatik Spektrum, number 15, pages 195-207.
- [2018 PYPL POLULARITY OF PROGRAMMING LANGUAGE]: 2018; PYPL POLULARITY OF PROGRAMMING LANGUAGE; PYPL Index; http://pypl.github.io/PYPL.html.
- [2012 RAI and BECK]: 2012; Dovan RAI, Joseph E. BECK; Math learning environment with game-like elements: an incremental approach for enhancing student engagement and learning effectiveness; Conference Intelligent Tutoring Systems, Publisher Springer, pages 90-100, ISBN 3642309496.
- [2006 RAZZAQ and HEFFERNAN]: 2006; Leena RAZZAQ, Neil T. HEFFERNAN; Scaffolding vs. hints in the Assistment System; Intelligent Tutoring Systems, Springer Berlin, Heidelberg.
- [2009 RAZZAQ, PATVARCZKI and ALMEIDA]: 2009; Leena RAZZAQ, Jozsef PATVARCZKI, Shane F. ALMEIDA, Manasi VARTAK, Mingyu FENG, Neil T. HEFFERNAN, Kenneth R. KOEDINGER; The Assistment Builder: Supporting the life cycle of tutoring system content creation; IEEE Transactions on Learning Technologies 2, number 2, pages 157-166..
- [2010 RAZZAQ and HEFFERNAN]: 2010; Leena RAZZAQ, Neil T. HEFFERNAN; Hints: is it better to give or wait to be asked?; Conference International Conference on Intelligent Tutoring Systems, Publisher Springer, pages 349-358.
- [1992 RECHTIN]: 1992; Eberhard RECHTIN; The art of systems architecting; IEEE Spectrum 29, number 10, pages 66-69.
- [2018 REDMONK]: 2018; REDMONK; The RedMonk Programming Language Rankings: January 2018; http://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/.
- [2001 REISER]: 2001; Robert A. REISER; A history of instructional design and technology: Part I: A history of instructional media; Journal Educational technology research and development, volume 49, issue 1, pages 53-64, ISSN 1042-1629.
- [2000 RIEHLE]: 2000; Dirk RIEHLE; Framework design: A role modeling approach; PHD thesis, ETH Zurich.
- [2005 RODRIGUES, NOVAIS and SANTOS]: 2005; Manuel RODRIGUES, Paulo NOVAIS, Manuel Filipe SANTOS; Future challenges in intelligent tutoring systems: a framework; Recent Research Developments in Learning Technologies.

- [2000 ROSATELLI, SELF and THIRY]: 2000; Marta ROSATELLI, John A. SELF, Marcello THIRY; LeCs: A collaborative case study system; International Conference on Intelligent Tutoring Systems, pages 242-251, Springer Verlag, Berlin, Heidelberg.
- [1957 ROTH]: 1957; Heinrich ROTH; Pädagogische Psychologie des Lehrens und Lernens; Book, German, 7th edition, Hermann Schroedel Verlag.
- [1995 SCHANDA]: 1995; Franz SCHANDA; Computer-Lernprogramme; Weinheim: Beltz, number 5, pages 32-41.
- [1997 SCHMIDT]: 1997; Douglas C. SCHMIDT; Applying patterns and frameworks to develop object-oriented communication software; Handbook of programming languages.
- [2007 SCHULMEISTER]: 2007; Rolf SCHULMEISTER; Grundlagen Hypermedialer Lernsysteme; Oldenburg Verlag.
- [1998 SELF]: 1998; John SELF; The defining characteristics of intelligent tutoring systems research: ITSs care, precisely; International Journal of Artificial Intelligence in Education (IJAIED) 10, pages 350-364.
- [2018 SHARPENED PRODUCTIONS]: 2018; SHARPENED PRODUCTIONS; Framework Definition; http://techterms.com/definition/framework.
- [1999 SHAW, GANESHAN and JOHNSON]: 1999; Erin SHAW, Rajaram GANESHAN, W. Lewis JOHNSON, Douglas MILLAR; Building a case for agent-assisted learning as a catalyst for curriculum reform in medical education; Proceedings of the International Conference on Artificial Intelligence in Education, pages 509-516.
- [2001 SHELBY, SCHULZE and TREACY]: 2001; R. N. SHELBY, K. G. SCHULZE, D. J. TREACY, M. C. WINTERGILL, Kurt VAN LEHN, A. WEINSTEIN; An assessment of the Andes tutor; NAVAL ACADEMY ANNAPOLIS MD.
- [1991 SHUTE]: 1991; Valerie J. SHUTE; Rose garden promises of intelligent tutoring systems: Blossom or thorn; NASA Technical Reports.
- [1994 SHUTE and PSOTKA]: 1994; Valerie J. SHUTE, Joseph PSOTKA; Intelligent Tutoring Systems: Past, Present, and Future; Institution: DTIC Document.
- [2017 SHYLESH]: 2017; S. SHYLESH; A Study of Software Development Life Cycle Process Models; National Conference on Reinventing Opportunities in Management, IT, and Social Sciences, pages 534-541.
- [2005 SIDNEY, CRAIG and GHOLSON]: 2005; K. Dmello SIDNEY, Scott D. CRAIG, Barry GHOLSON, Stan FRANKLIN, Rosalind PICARD, Arthur C. GRAESSER; Integrating affect sensors in an intelligent tutoring system; Affective Interactions: The Computer in the Affective Loop Workshop, pages 7-13.
- [1998 SIEMER and ANGELIDES]: 1998; Julika SIEMER, Marios C. ANGELIDES; A comprehensive method for evaluation of complete intelligent tutoring systems; Decision support systems 22, number 1, pages 85-102.
- [2011 SINGH, SALEEM and PRADHAN]: 2011; Ravi SINGH, Muhammad SALEEM, Prabodha PRADHAN, Cristina HEFFERNAN, Neil T. HEFFERNAN, Leena RAZZAQ, Matthew D. DAILEY, Cristine O'CONNOR; Feedback during web-based homework: the role of hints; International Conference on Artificial Intelligence in Education, Springer, Berlin, Heidelberg.
- [1982 SLEEMAN and BROWN]: 1982; Derek H. SLEEMAN, John Seely BROWN; Intelligent Tutoring Systems; Book, London Academic Press, 345 pages, HAL Id hal-00702997, version 1.
- [1997 SONG, HAHN and TAK]: 1997; J. S. SONG, S. H. HAHN, K. Y. TAK, J. H. KIM; An intelligent tutoring system for introductory C language course; Computers in Education, Computers & education 28, number 2, pages 93-102.

- [1995 SONI, NORD and HOFMEISTER]: 1995; Dilip SONI, Robert L. NORD, Christine HOFMEISTER; Software architecture in industrial applications; Software Engineering, 1995. ICSE 1995. 17th International Conference on, IEEE, pages 196-196.
- [2011 SOUHAIB, MOHAMED and KADIRI KAMAL EDDINE]: 2011; Aammou SOUHAIB, Khaldi MOHAMED, El KADIRI KAMAL EDDINE; Architectural Models of Adaptive Hypermedia Based on the Use of Ontologies; Online Submission, US-China Education Review A 3 p297-306.
- [2018 SQUAD and TEACHERGAMING]: 2018; SQUAD, TEACHERGAMING; Kerbal Space Program/KerbalEDU; https://kerbaledu.com.
- [1996 STANKOV]: 1996; Slavomir STANKOV; Computers in Education: Technological Transformation; Journal Development and Perspectives, GRKG Humankybernetik, volume 37, number 1, pages 16-26.
- [1984 STRZELEWICZ]: 1984; Willy STRZELEWICZ; Lebenslanges Lernen als Bildungsaufgabe in sozialhistorischer Sicht; Erwachsenenbildung als Wissenschaft, Akten des Kongresses der Weltenburger Akademie, number 12, pages 29–53.
- [2008 SUAREZ and SISON]: 2008; Merlin SUAREZ, Raymund SISON; Automatic construction of a bug library for object-oriented novice java programmer errors; Intelligent Tutoring Systems, Springer Berlin Heidelberg.
- [2002 SURAWEERA AND ANTONIJA MITROVIC and MITROVIC]: 2002; Pramuditha SURAWEERA AND ANTONIJA MITROVIC, Antonija MITROVIC; KERMIT: A constraint-based tutor for database modeling; Intelligent Tutoring Systems, Publisher Springer Verlag, pages 377-387, ISBN/ISSN 3540437509.
- [2018 SWIFT MANAGEMENT AG]: 2018; SWIFT MANAGEMENT AG; iTheorie; https://www.itheorie.ch.
- [2018 THE COMPUTER LANGUAGE BENCHMARKS GAME]: 2018; THE COMPUTER LANGUAGE BENCHMARKS GAME; all Swift programs & measurements; https://benchmarksgame.alioth.debian.org/u64q/measurements.php?lang=swift.
- [2000 THIBODEAU, BÉLANGER and FRASSON]: 2000; Marc-André THIBODEAU, Simon BÉLANGER, Claude FRASSON; WHITE RABBIT-Matchmaking of user profiles based on discussion analysis using intelligent agents; International Conference on Intelligent Tutoring Systems, pages 113-122, Springer Verlag, Berlin, Heidelberg.
- [2009 TIDDLYSPOT.COM]: 2009; TIDDLYSPOT.COM; Programmed Instruction examples; http://programmedinstruction.tiddlyspot.com.
- [2012 TILMANN, MOSKAL and DE HALLEUX]: 2012; Nikolai TILMANN, Michal MOSKAL, Jonathan DE HALLEUX, Manuel FAHNDRICH, Judith BISHOP, Arjmand SAMUEL, Tao XIE; The future of teaching programming is on mobile devices; Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education.
- [2017 TIOBE.COM]: 2017; TIOBE.COM; TIOBE-Index; http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
- [2014 TRIPATHY and NAIK]: 2014; Priyadarshi TRIPATHY, Kshirasagar NAIK; Software Evolution and Maintenance: A Practitioner's Approach; Book, John Wiley & Sons.
- [2010 TSOVALTZI, MCLAREN and MELIS]: 2010; Dimitra TSOVALTZI, Bruce M. MCLAREN, Erica MELIS, Ann-Kristin MEYER, Micheal DIETRICH, George GOGUADZE; Learning from erroneous examples; International Conference on Intelligent Tutoring Systems, pages 420-422, Springer Berlin Heidelberg.

- [1969 UHR]: 1969; Leonard UHR; Teaching machine programs that generate problems as a function of interaction with students; Proceedings of the 1969 24th national conference, Publisher ACM, pages 125-134, DOI 10.1145/800195.805924.
- [2016 UNESCO]: 2016; UNESCO; The World Needs Almost 69 Million New Teachers to Reach the 2030 Education Goals; UNESCO Institute for Statistics http://uis.unesco.org/en/documents.
- [2000 UNIVERSITÄT HANNOVER]: 2000; UNIVERSITÄT HANNOVER; Abschlussbericht des Projektverbundes "Virtueller Campus"; Universitäten Hannover, Hildesheim, Osnabrück.
- [1996 URBAN-LURAIN]: 1996; Mark URBAN-LURAIN; Intelligent tutoring systems: An historic review in the context of the development of artificial intelligence and educational psychology; Michigan State University, ITS Annual Review.
- [1976 VAN HOUT and METTES]: 1976; J. F. H. J. VAN HOUT, C. T. C. W. METTES; The effect of egruleg versus ruleg and teacher-centredness versus student-centredness on pupil gain and satisfaction
- ; Journal Instructional science 5, number 2, pages 181-187, Harvard, EISSN: 15731952, ISSN: 00204277.
- [2011 VAN LEHN]: 2011; Kurt VAN LEHN; The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems; Educational Psychologist 46.4, pages 197-221.
- [2018 VARSITY TUTORS]: 2018; VARSITY TUTORS; AplusMath; https://www.varsitytutors.com/aplusmath.
- [1995 VASANDANI and GOVINDARAJ]: 1995; Vijay VASANDANI, T. GOVINDARAJ; Knowledge organization in intelligent tutoring systems for diagnostic problem solving in complex dynamic domains; IEEE Transactions on Systems, Man, and Cybernetics 25, number 7, pages 1076-1096.
- [2019 VAWTER, GRAF VON MALOTKY and MARTENS]: 2019; Laura VAWTER, Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; The implications of ITS research on CALL software; 11th annual International Conference on Education and New Learning Technologies (EDULEARN), pages, 1562-1568, DOI 10.21125/edulearn.2019.0463.
- [2000 VIRVOU and MOUNDRIDOU]: 2000; Maria VIRVOU, Maria MOUNDRIDOU; Modelling the instructor in a Web-based authoring tool for Algebra-related ITSs; International Conference on Intelligent Tutoring Systems, pages 635-644, Springer Verlag, Berlin, Heidelberg.
- [2000 VIZCAÍNO, CONTRERAS and FAVELA]: 2000; Aurora VIZCAÍNO, Juan CONTRERAS, Jesús FAVELA, Manuel PRIETO; An adaptive, collaborative environment to develop good habits in programming; International Conference on Intelligent Tutoring Systems, pages 262-271, Springer Verlag, Berlin, Heidelberg.
- [2006 WALKER, KOEDINGER and MCLAREN]: 2006; Erin WALKER, Kenneth KOEDINGER, Bruce MCLAREN; Cognitive tutors as research platforms: Extending an established tutoring system for collaborative and metacognitive experimentation; International Conference on Intelligent Tutoring Systems, pages 207-216, Springer Verlag, Berlin, Heidelberg.
- [2006 WALONOSKI]: 2006; Jason A. WALONOSKI; Detection and analysis of off-task gaming behavior in intelligent tutoring systems; International Conference on Intelligent Tutoring Systems, pages 382-391, Springer.
- [2019 WASSMANN, GRAF VON MALOTKY and MARTENS]: 2019; Ingolf WASSMANN, Nikolaj Troels GRAF VON MALOTKY, Alke MARTENS; Train4U - Mobile Sport Diagnostic Expert System for User-Adaptive Training; Proceedings of the 12th International Sympo-

sium on Computer Science in Sport (IACSS), Advances in Intelligent Systems and Computing, volume 1028, Springer, Cham, pages 77-85.

- [2018 WE ❤ SWIFT]: 2018; WE ❤ SWIFT; WeHeartSwift; https://www.weheartswift.com.
- [2018 WEEBLY.COM]: 2018; WEEBLY.COM; The Baby Boom of the 1950s; https://1950s.weebly.com/the-baby-boom.html.
- [2006 WEI and BLANK]: 2006; Fang WEI, Glenn D. BLANK; Student modeling with atomic bayesian networks; International Conference on Intelligent Tutoring Systems, pages 491-502, Springer, Berlin, Heidelberg.
- [1987 WENGER]: 1987; Etienne WENGER; Intelligent Tutoring Systems; Morgan Kaufmann, Menlo Park, CA.
- [1987 WENGER]: 1987; Etienne WENGER; Artificial intelligence and tutoring system: Computational and Cognitive Approaches to the Communication of Knowledge; Texto publicado na: Pátio-revista pedagógica Editora Artes Médicas Sul Ano 1, pages 19-21, Califórnia, Morgan Kaufmann Publishers.
- [1990 WERTHEIMER]: 1990; Richard WERTHEIMER; The geometry proof tutor: An" intelligent" computer-based tutor in the classroom"; The Mathematics Teacher 83, number 4, pages 308-317.
- [2018 WEWANTTOKNOW AS]: 2018; WEWANTTOKNOW AS; DragonBox Algebra 12+; https://dragonbox.com/products/algebra-12.
- [2000 WIGGINS and TREWIN]: 2000; Geraint A. WIGGINS, Shari TREWIN; A system for concerned teaching of musical aural skills; International Conference on Intelligent Tutoring Systems, pages 494-503, Springer Verlag, Berlin, Heidelberg.
- [2018 WIKIA.COM]: 2018; WIKIA.COM; Video game industry; http://vgsales.wikia.com/wiki/Video_game_industry.
- [2011 WILLIAM and SURI]: 2011; Jack WILLIAM, Tavneet SURI; Mobile money: The economics of M-PESA; National Bureau of Economic Research, number w16721.
- [2010 WOOLF]: 2010; Beverly Park WOOLF; Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning; Publisher Morgan Kaufmann, ISBN 0080920047.
- [2018 XU, ZHOU and WANG]: 2018; Tao XU, Yun ZHOU, Zi WANG, Yixin PENG; Learning emotions EEG-based recognition and brain activity: A survey study on BCI for intelligent tutoring system; Procedia computer science 130, pages 376-382.
- [2002 YANG and PATEL]: 2002; Ang YANG, Ashok PATEL; A Plugable Web Based Intelligent Tutoring System; ECIS Proceedings, page 100.
- [1975 YOURDON]: 1975; Edward YOURDON; Structured programming and structured design as art forms; Proceedings of the National Computer Conference and Exposition on - AFIPS, page 277, DOI 10.1145/1499949.1499997.
- [2007 ZINS]: 2007; Chaim ZINS; Conceptual approaches for defining data, information, and knowledge; Journal of the American society for information science and technology, 58(4), 479-493.
- [2000 ZOUAQ, FRASSON and ROUANE]: 2000; Amal ZOUAQ, Claude FRASSON, Khalid ROUANE; The explanation agent; International Conference on Intelligent Tutoring Systems, pages 554-563, DOI 10.1007/3-540-45108-0_59, Springer Verlag, Berlin, Heidelberg.