

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



**FEUP**

**Software Knowledge Management  
using Wikis: a plugin for weakly typed  
pages**

**Michał Kacprzyk**

Dissertation

Master in Informatics and Computing Engineering

Supervisor: Ademar Aguiar (PhD.)

25<sup>th</sup> March, 2010



# **Software Knowledge Management using Wikis: a plugin for weakly typed pages**

**Michał Kacprzyk**

Dissertation

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Maria Cristina Ribeiro (PhD.)

---

External Examiner: Isabel Ramos - DSI/UMinho (PhD.)

Internal Examiner: Ademar Aguiar (PhD.)

25<sup>th</sup> March, 2010



# Abstract

This thesis addresses the topic of software knowledge management using wikis.

In 2010 virtually all Internet users know at least one wiki - Wikipedia - "the largest and most popular general reference work on the Internet". The example of Wikipedia shows that wikis are a very good collaborative tools. In fact wikis are widely used around the world in various applications, also in software engineering companies.

The software engineering was introduced because of burning need to create better quality software. Hardware evolution was very fast and informal software development was not sufficient any more. The software engineering consists of sets of activities and models, which covers whole life-cycle of the software. There is a lot of knowledge involved in software engineering. Basically, creation of software is about incrementally transforming and formalizing knowledge - from the idea of needed application into the specific source code. Usually, there is many people collaborating on the particular software and they need to exchange the knowledge many times. They also need to know the new technologies, common practices, policy of the firm etc.

Because software engineering is knowledge intensive activity, any tool that can help to manage the software knowledge will be also helpful in improving software engineering activities. In order to find that tools, we can look into the modern discipline called knowledge management. Besides of giving general awareness of knowledge related processes, it brings set of practical tools useful to improve organization's knowledge infrastructure. It can be generally divided into two knowledge management strategies: codification, to capture and store explicit knowledge, and personalization, to improve tacit knowledge sharing through improving connectivity between people.

When we connect the collaborative power of wikis, knowledge intensive software engineering activities and knowledge management, we will get an interesting combination. Although wikis are already useful tools, they can be improved even further.

One of the problems I have encountered in the past during application of wiki in the software engineering company was struggle between freedom of creating wiki pages and formality of software engineering. One of the possible solutions is the idea of wiki pages loosely connected to page types. Creation of the prototype of this solution is described step by step, from the idea and design into implementation and validation. It can be useful to add formalized structure into wiki without losing the openness and flexibility.



*“All that we are is the result of what we have thought.  
The mind is everything.  
What we think, we become.”*

— Buddha





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Outline . . . . .	3
<b>2</b>	<b>The theory behind the subject</b>	<b>5</b>
2.1	Software engineering . . . . .	5
2.1.1	Software process activities . . . . .	6
2.1.2	Software process models . . . . .	7
2.2	Knowledge management . . . . .	15
2.2.1	Knowledge . . . . .	16
2.2.2	Tacit and Explicit Knowledge . . . . .	18
2.2.3	Knowledge management Life Cycle . . . . .	21
2.3	Software knowledge management . . . . .	22
2.3.1	Knowledge management strategies and generations . . . . .	24
2.3.2	Post-Mortem Analysis . . . . .	25
2.4	Tools for software knowledge management . . . . .	26
2.4.1	Wikis . . . . .	29
2.4.2	Semantic wikis . . . . .	32
<b>3</b>	<b>WeakType plugin</b>	<b>37</b>
3.1	WeakType idea . . . . .	37
3.2	Creation of the plugin . . . . .	38
3.2.1	Specification . . . . .	38
3.2.2	Architectural design . . . . .	40
3.2.3	Interface design . . . . .	42
3.2.4	Component design . . . . .	44
3.2.5	Data structure design . . . . .	46
3.2.6	Algorithm design . . . . .	48
3.2.7	Implementation . . . . .	51
<b>4</b>	<b>Validation and Use-Cases</b>	<b>53</b>
4.1	Scenario 1 - Web development . . . . .	53
4.1.1	Use-Case 1 . . . . .	54
4.1.2	Use-Case 2 . . . . .	55
4.2	Scenario 2 - Migration of the documentation . . . . .	60

## CONTENTS

<b>5 Conclusion</b>	<b>63</b>
5.1 Results . . . . .	63
5.2 Future work . . . . .	63
<b>References</b>	<b>66</b>

# List of Figures

1.1	Chosen significant knowledge sharing related events. . . . .	2
2.1	The waterfall model. . . . .	8
2.2	Evolutionary development. . . . .	9
2.3	Component-based software engineering. . . . .	10
2.4	Incremental delivery. . . . .	11
2.5	Spiral development. . . . .	12
2.6	The Rational Unified Process. . . . .	14
2.7	Relation of data, information, knowledge and wisdom. . . . .	17
2.8	Tacit and explicit knowledge[ <a href="#">NT95</a> ]. . . . .	19
2.9	Knowledge management Life Cycle[ <a href="#">SAA99</a> ] . . . . .	21
2.10	General map of knowledge management tasks. . . . .	23
2.11	Wiki Wikipedia page. . . . .	30
2.12	RDF triple . . . . .	33
2.13	Albert Einstein page’s categories in Wikipedia. . . . .	34
3.1	Base wiki page. . . . .	38
3.2	Type page. . . . .	39
3.3	Base wiki page with WeakType plugin data. . . . .	39
3.4	Result page. . . . .	40
3.5	General view of whole process. . . . .	41
3.6	Architectural design. . . . .	42
3.7	Relations between interface elements. . . . .	43
3.8	States of the syntax component. . . . .	44
3.9	States of the action component. . . . .	45
3.10	Source code of the headers in DokuWiki. . . . .	46
3.11	Structure of this chapter as a DokuWiki headers. . . . .	46
3.12	Structure of this chapter as tree of headers. . . . .	47
3.13	Pass 1: Get match and not match base lines. . . . .	49
3.14	Pass 2: Recreate matching array. . . . .	50
4.1	Brief from the meeting with client. . . . .	56
4.2	Brief with the structure of ”bussines card” type. Graphics adapted from [ <a href="#">Kam09</a> ]. . . . .	57
4.3	Bussines card page converted into portal type structure. . . . .	58
4.4	Finished brief of portal page. . . . .	59
4.5	Example of not unified documentation page. . . . .	61
4.6	Page with the structure of ”JCL” type. . . . .	61

## LIST OF FIGURES

4.7	Improved structured documentation page. . . . .	62
5.1	Idea of interface improvement. . . . .	64

# Definitions and Abbreviations

- AD** Anno Domini
- API** Application Programming Interface
- BC** Before Christ
- CASE** Computer Aided Software Engineering
- CICS** Customer Information Control System
- CMS** Content Management System
- COBOL** COmmon Business Oriented Language
- DB** DataBase
- FTP** File Transfer Protocol
- GNU** GNU's Not Unix
- GPL** General Public License
- GUI** Graphical User Interface
- HTML** HyperText Markup Language
- HTTP** Hypertext Transport Protocol
- IBM** International Business Machines
- IDE** Integrated Development Environment
- JCL** Job Control Language
- KM** Knowledge Management
- MOSS** Microsoft Office Share Point Server
- MS** Microsoft
- NATO** North Atlantic Treaty Organization
- PC** Personal Computer
- PHP** PHP Hypertext Preprocessor

## DEFINITIONS AND ABBREVIATIONS

- PMA** Post-Mortem Analysis
- RDF** Resource Description Framework
- REXX** REstructured eXtended eXecutor
- RUP** Rational Unified Process
- SE** Software Engineering
- SPARQL** SPARQL Protocol And RDF Query Language
- SVN** Subversion
- TFS** Team Foundation Server
- UML** Universal Modelling Language
- URI** Uniform Resource Identifier
- URL** Uniform Resource Locator
- USB** Universal Serial Bus
- W3C** World Wide Web Consortium
- WSS** Windows Share Point Services
- XHTML** Extensible HyperText Markup Language
- XML** Extensible Markup Language
- XMPP** Extensible Messaging and Presence Protocol

# Chapter 1

## Introduction

”Egyptian hieroglyphs are generally considered the earliest writing systems it emerged about 3400-3200 BC.”<sup>1</sup>. ”The immediate predecessor to modern paper is believed to have originated in China in approximately the 2nd century AD.”<sup>2</sup> ”The age of printed book started in 1450s by printing Gutenberg Bible.”<sup>3</sup>”In 1936 Alan Turing provided an influential formalisation of the concept of the algorithm and computation with the Turing machine.”<sup>4</sup> ”The term software engineering first appeared in the 1968 NATO Software Engineering Conference and was meant to provoke thought regarding the current ”software crisis” at the time.”<sup>5</sup> ”The IBM Personal Computer, commonly known as the IBM PC, is the original version and progenitor of the IBM PC compatible hardware platform. It is IBM model number 5150, and was introduced on August 12, 1981.”<sup>6</sup> ”WikiWikiWeb is a term that has been used to refer to four things: the first wiki, or user-editable website, launched in 1995 by Ward Cunningham.”<sup>7</sup> ”Wikipedia was launched in 2001 by Jimmy Wales and Larry Sanger and is currently the largest and most popular general reference work on the Internet”.<sup>8</sup>

All information in the first paragraph are taken directly from the Wikipedia - ”the largest and most popular general reference work on the Internet”. It is at least amazing that people from the level of hieroglyphs came to the tool like Wikipedia. When we look at the figure 1.1 we will see the time line divided by some key events in the evolution of ways of sharing knowledge. It is clearly visible that this process progresses rapidly. When we consider that 10 years ago, there was no Wikipedia, it is impossible to predict what will come in the next 10 years. However, we can identify major trends - growing of knowledge repositories and improving ways of sharing knowledge.

---

<sup>1</sup>[http://en.wikipedia.org/wiki/Writing\\_system](http://en.wikipedia.org/wiki/Writing_system)

<sup>2</sup><http://en.wikipedia.org/wiki/Paper>

<sup>3</sup>[http://en.wikipedia.org/wiki/Gutenberg\\_Bible](http://en.wikipedia.org/wiki/Gutenberg_Bible)

<sup>4</sup><http://en.wikipedia.org/wiki/Computer>

<sup>5</sup>[http://en.wikipedia.org/wiki/Software\\_engineering](http://en.wikipedia.org/wiki/Software_engineering)

<sup>6</sup>[http://en.wikipedia.org/wiki/IBM\\_PC](http://en.wikipedia.org/wiki/IBM_PC)

<sup>7</sup><http://en.wikipedia.org/wiki/Wikiwikiweb>

<sup>8</sup><http://en.wikipedia.org/wiki/Wikipedia>

## INTRODUCTION

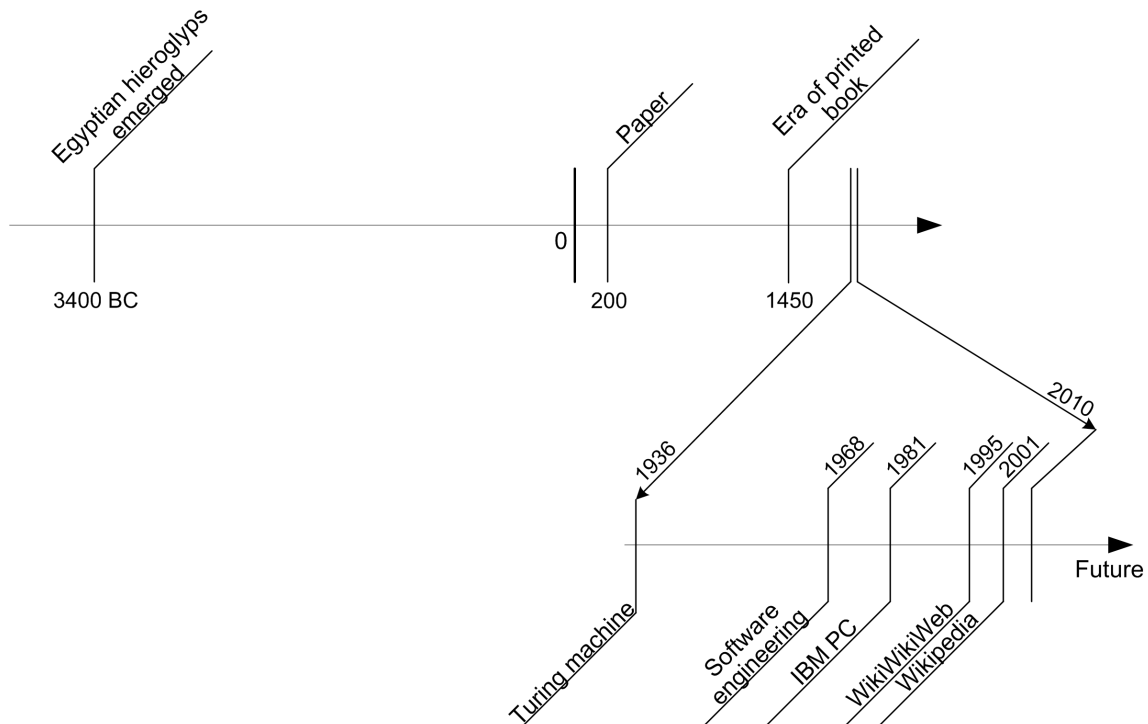


Figure 1.1: Chosen significant knowledge sharing related events.

Software engineering comes from burning need to create good quality software. Process of software creation is often called software development and it can be compared to building a house. In the beginning there is only an idea of having a new house. It is fuzzy and very general, but it can already have some specifications like: "I want to have a big kitchen and two bedrooms". During the process of thinking about it, bit by bit, new ideas come to mind: "It should be located in the quiet countryside, but not too far from the city. It has to be warm in the winter and cool in the summer". When we have quite sufficient for us specification, we can go to the architect for the design. The design can be based on the existing one or build from the scratch. After having general design of number and placement of rooms, number of floors, occupied surface etc., there is a time to think of all other, more specific things... At some level of designing, we start actually building a house, now we need a lot of social and management skills to hire right people and coordinate them properly. Of course, we can not forget about a budget and a time. But, in many cases this is not the end of designing, now more specific things come to play: type of building materials, heating, roof surface, windows, placement of electric sockets and lamps. Later, even more specific things like a kind of floor and wall surface, bathroom equipment become to be important. It takes really a lot of effort to come to the moment, when you can paint the walls with your favourite colors and buy furniture. Hopefully, there is a moment when finally everything is finished and you can move in.

This metaphoric process shows that software creation is complex, containing many



## INTRODUCTION

steps and it needs knowledge from many sources. The knowledge is gathered and stored incrementally during the process of creation. Like in the creating a painting, it starts from first line, sketch, first layer of paint, second layer... It is very similar to the process of creation of a wiki page.

We can investigate the evolution of Albert Einstein page in English Wikipedia, using revision history. First revision was created on 5 November 2001, it consisted of 8036 chars and no pictures or tables. In November 2003 about 300th revision of the page was available which consisted of 22 666 chars and one big picture. In November 2005 it was about in 2500th revision which consisted of 42 289 chars, 8 pictures and several tables, data becomes organized in sections. Up to now - the time of writing this thesis, about 13 000th revision of the page is available. It is consisted of 87 061 chars, many tables and pictures.

That example shows that wikis are a good tool to gather and systematize incrementally growing knowledge. As we already know, the process of development is about combining and converting knowledge into the final product. Although, wikis are already very useful in software development, they still can be improved. In the journey to build better knowledge related tools, nowadays, using wikis as a base can be seen as "standing on the shoulders of giants". In this thesis it is expected to see wikis in the wider context and try to implement one of the wiki's improvement ideas.

### **1.1 Outline**

The thesis is organized into 5 chapters.

#### **Chapter 1 - Introduction**

Introduces into the thesis, shows context of the work, motivation and expected results.

#### **Chapter 2 - The theory behind the subject**

It is a review of actual chosen literature and conceptions about the subject. It is based on the academic research, white papers and well-known books. It is subdivided into 4 sub-chapters.

#### **Chapter 2.1 - Software engineering**

Introduces general concepts of Software Engineering. Describes origin and purpose of existence of such field of science. Shows general concerns of the field such as steps of software process activities. Then, it presents several software process models from traditional waterfall model to modern Rational Unified Process.

## INTRODUCTION

### **Chapter 2.2 - Knowledge management**

Introduces general concepts of knowledge management. It starts by defining key objects of KM interest - data, information and knowledge. Later, it describes popular conception of tacit and explicit knowledge and ways of transforming it from one to another. At the end, it describes example of knowledge management life cycle. Because field of knowledge management often use indirect ways of sharing knowledge like stories, metaphors and proverbs, this sub-chapter is written in a similar style.

### **Chapter 2.3 - Software knowledge management**

This sub-chapter connects two previous ones. It shows concepts of "organizational learning" and "learning software organization" and enumerates knowledge needs of software engineering companies. Later, it divides knowledge management into generations and shows general strategies, particularly used in software knowledge management - codification and personalization. At the end, it shows one simple, but very useful tool from knowledge management - Post-Mortem Analysis.

### **Chapter 2.4 - Tools for software knowledge management**

Firstly, it treats about tools used in the particular steps of software creation process. It distinguishes tools useful on the level of project from management tools. It briefly describes enterprise software knowledge management tools from main vendors like Microsoft and IBM, also shows open source alternatives. Secondly, in more details wikis and semantic wikis are described.

### **Chapter 3 - WeakType plugin**

This chapter shows a creation process of the tool - practical part of thesis. Firstly, it introduces the idea of weak type wiki pages. Secondly, it shows step by step, the creation process of the proposed solution. The steps are: specification, architectural design, interface design, data structure design, algorithm design and implementation.

### **Chapter 4 Validate and case studies**

This chapter shows examples of practical applications of the created prototype. It is divided into two scenarios. First describes usage in a web development company as a tool to support requirements gathering process. In the second scenario, prototype is used to unify the structure of the system documentation.

### **Chapter 5 Conclusion**

Final chapter concludes all the work done. It describes results and gives ideas for the future work.

## Chapter 2

# The theory behind the subject

### 2.1 Software engineering

This chapter is going to sketch general ideas about software engineering. After short introduction, the main software engineering activities and models are briefly described.

The term software engineering was invented in 1968 during the NATO conference about "software crisis"[NR69]. The reason of this crisis was the introduction of new computer hardware which allowed to build much bigger and complex software. It became clear that informal software development was not sufficient. New techniques and methods were needed. These techniques become part of software engineering.

The definition says:

"Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software"[IEE04]

Software engineering has many linkages with other fields of science. One obvious connection is with computer science since software is dedicated for computers. Next connection which is strongly connected to the first one is with physics, because physics is the foundation of today's electronic computers. Passes through the electronics, because of the types of processors, memory, input and output devices to which it has to be adapted. Draws abundantly from the "queen of sciences"<sup>1</sup> mathematics, for example in areas such as algorithms optimization. On the other hand, psychology, because the software is created by the people and often for the people as the end user. Worth to point here is the fact that at present only a human can create software. It is not possible to create software by computers only, but of course without computers probably software engineering

---

<sup>1</sup>Carl Friedrich Gauss referred to mathematics as "the Queen of the Sciences"

## THE THEORY BEHIND THE SUBJECT

would not be ever invented. Software engineering due to the rapid spread of computers and networks is also related to sociology.

Currently, the software serves people in virtually all areas of life. Specialists, researchers and all the average office workers uses it every day to improve the efficiency and quality of their work. The software allows to solve problems in which use of traditional piece of paper and a pencil did not give sufficient results in a finite time. In personal life, it can provide us an entertainment with computer games, whether to allow to make specific holiday plan. Lets us communicate faster, easier and with more people simultaneously. Enhances discovery, selection and sharing informations. We can find countless number of examples which shows how software is useful and needed now-days. It is clear that the modern technologies which includes billions of lines of code are present everywhere. It is very difficult today to imagine how would our lives look like if the software does not exist at all. Once we realized the magnitude of the subject, we can briefly consider what would happen if the software was of poor quality or created with significant delay. We can easily imagine the possible tragedies which may happen if the software dedicated for “critical system” such as air traffic control system would have been full of errors.

### **2.1.1 Software process activities**

From global point of view, software development is a process of solving, often sophisticated problems with knowledge, effort and tools. In almost every software process there are few common characteristic activities:

#### **Software specification**

During this activity the functionality of the system is decided. “Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system’s operation and development.”[Som07] This activity can be divided into four phases:

1. Feasibility study
2. Requirements elicitation and analysis
3. Requirements specification
4. Requirements validation

#### **Software design**

In this activity the solution for the system is developed, software architecture is decided, design patterns are chosen. As a result detailed design is created. “A software design is a

## THE THEORY BEHIND THE SUBJECT

description of the structure of the software to be implemented(...).”[Som07] This process can be divided into six parts:

1. Architectural design
2. Abstract specification
3. Interface design
4. Component design
5. Data structure design
6. Algorithm design

### **Software implementation**

Now previously created design is implemented. The reusable software and implementation patterns can be used. “The implementation stage of software development is the process of covering a system specification into an executable system.”

### **Software verification and validation**

This activity is done to make sure that the software meets specification and customer’s needs.”(...)verification and validation is intended to show that a system conforms to its specification and that the system meets the expectations of the customer buying the system.”[Som07] One important part of verification and validation is testing. Testing process can be divided into:

1. Basic/Unit Test
2. Functional/Integration Test
3. System Verification
4. Acceptance Test

### **Software evolution**

The system changed to meet changing customer needs.“The flexibility of software systems is one of the main reasons why more and more software is being incorporated in large, complex systems.”[Som07]

### **2.1.2 Software process models**

In software engineering there is many software process models designed to improve software development. Few main models are presented below:

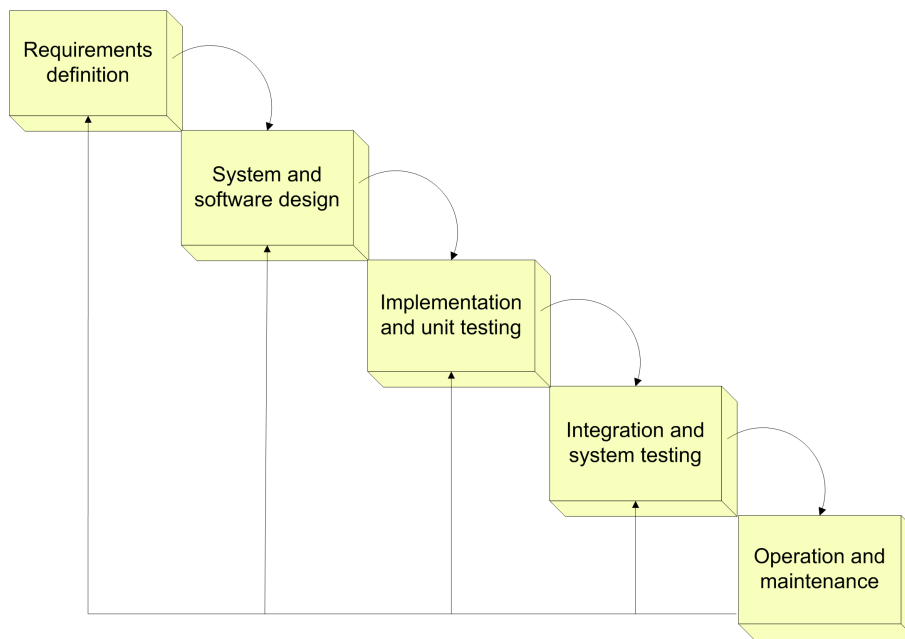


Figure 2.1: The waterfall model.

### The waterfall model

In this model the fundamental process activities such as specification, development, validation and evolution are represented in separate process phases: requirements specification, software design, implementation, testing etc. It is the first model of software development process which was published. As we can see in the figure 2.1 the model contains five main phases:

**Requirements definition** System's functionality, constraints and aims are decided during consultations with system users. Later they are defined in detail and become system specification

**System and software design** Overall system architecture is established. In software design process the main system abstractions and their relations are decided and described.

**Implementation and unit testing** The program representation of software design is created. Normally it is a set of programs or program units. During unit testing each unit is tested and verified if it meets the specification.

**Integration and system testing** Program units or programs are integrated into complete system and testes as whole to make sure that the software meets requirements. After testing finished software is delivered to the client.

**Operation and maintenance** It is normally the longest life-cycle phase. The system is in practical use by end users. During maintenance correcting errors, which haven't

## THE THEORY BEHIND THE SUBJECT

been discovered earlier, takes place. In this phase also the software is improved to meet user's needs. In principal, every phase ends with one or more documents that are approved. In theory next phase should not start before previous one ends, but in practice often all phases overlap and feed information to each other. During design some problems with specification may appear, during implementation problems with design are discovered. "The software process is not simple linear model but involves a sequence of iterations of the development activities." [Som07]

### Evolutionary development

In this case the activities of specification, development and validation are interleaved. The main idea is to create the prototype rapidly from abstract specification and then expose it to the client and through many versions create the final system which will satisfy user's needs.

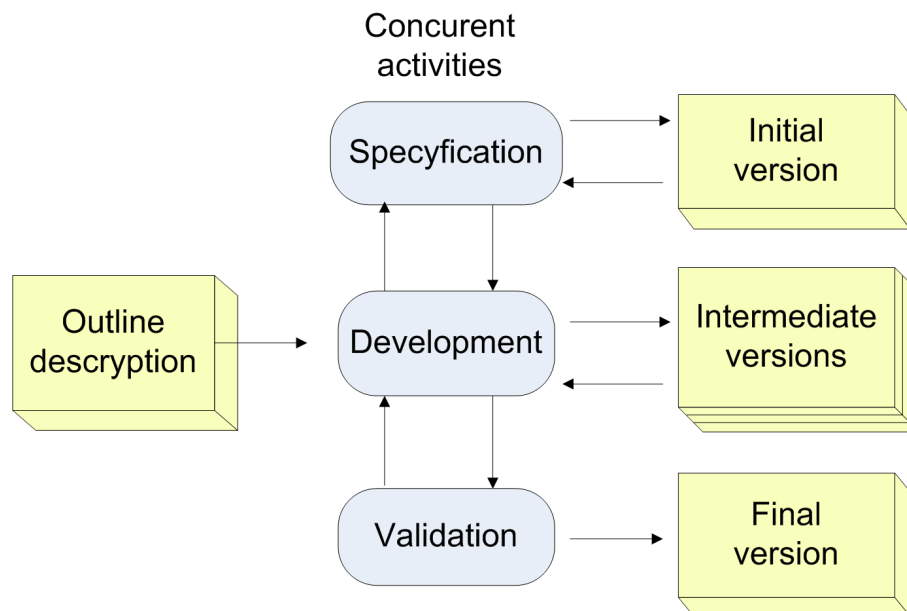


Figure 2.2: Evolutionary development.

Two types of evolutionary development are:

**Exploratory development** The development starts with the well understood parts of the system. The system grows by adding new features which the client proposes.

**Throwaway prototyping** The development starts with poorly understood requirements. Exposing the prototype to the client helps to better understand the requirements.

### Component-based software engineering

In this approach the significant part is use of reusable components. The development process is based on combining this components rather than creating the system from scratch.

## THE THEORY BEHIND THE SUBJECT

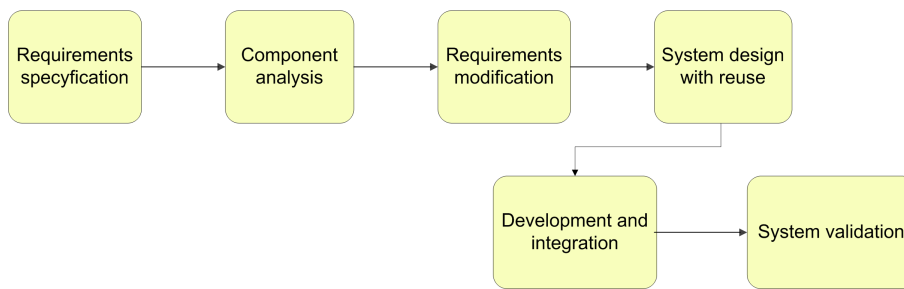


Figure 2.3: Component-based software engineering.

Software reuse takes place in many software projects. Usually it is informal. However, lately component-based software engineering has emerged and is widely used. This approach is based on a large set of reusable software components. In component-based software engineering specification and validation phases are similar to other processes. Differences appears in intermediate phases:

**Component analysis** In this stage components which may match to the requirements are found. Usually it is impossible to find components which match the requirements perfectly, found components provide only part of the required functionality.

**Requirements modification** In this stage the requirements are re-analyzed in context of information about found components. The requirements are modified so they can match to the available components. It is possible to go back to component analysis if it is not possible to change requirements sufficiently.

**System design with reuse** Design of the system framework is created or an existing framework is reused. The base for the design is the set of components which are reused. If reusable components are not available for some features of the system, it is needed to design new software component.

**Development and integration** The software which cannot be based on reused components is developed and integrated together with components to build new system.

The Component-based software engineering has many big advantages. First of all, use of already created components reduces time needed to create the software, as well as costs and risks. On the other hand reused components may not fit exactly the requirements. This may lead to create the system which not meets user's needs.

### Iterative development

The idea of the iterative process is that the specification is developed together with the software. The software is created in many iterations rather than in linear process, which may be seen as a more natural way of software development, since in normal life also we



do not solve the problem at once, but work on different parts of it separately. Models created to support iterative development are:

**Incremental delivery** It is a combination of waterfall and evolutionary model. In this model customers identify the services which should be provided by the software. They also decide which services are the most important and which are the least important to them. Delivering of the system is divided into defined number of delivery increments. The most important services are delivered first. When system increments are decided, detailed definition of the requirements for the service which should be delivered in first increment takes place. Then the increment is developed. Analysis of the requirements for further increments may take place during development of the previous one, but it is not possible to change the requirements for the actual increment. When the new increment is delivered to the customer, it is integrated with already existing parts so the functionality of the system evolves with every delivered incrementation.

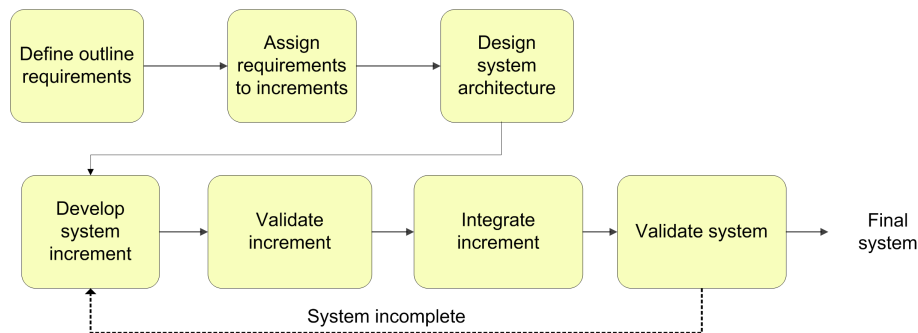


Figure 2.4: Incremental delivery.

**Spiral development** This model was introduced in 1986 by Barry Boehm.[Boe86] As the name tells, the software process is presented as a spiral, instead of a sequence of activities. Spiral is divided into a loops, where every loop is one phase of software process. In this model risks are consider as the main driver, so the risk management is crucial. As it can be seen on figure 2.5, each loop is split into four sectors:

- **Objective setting** - in this stage specific objects for the current loop are decided. The management plan is created. The risks are identified and the main strategies to handle risks are planned.
- **Risk assessment and reduction** - as it was stated before, risks are the main driver in this model. In this stage of the loop, detailed analyzing of every risk takes place and sufficient steps to reduce the risks are taken.

## THE THEORY BEHIND THE SUBJECT

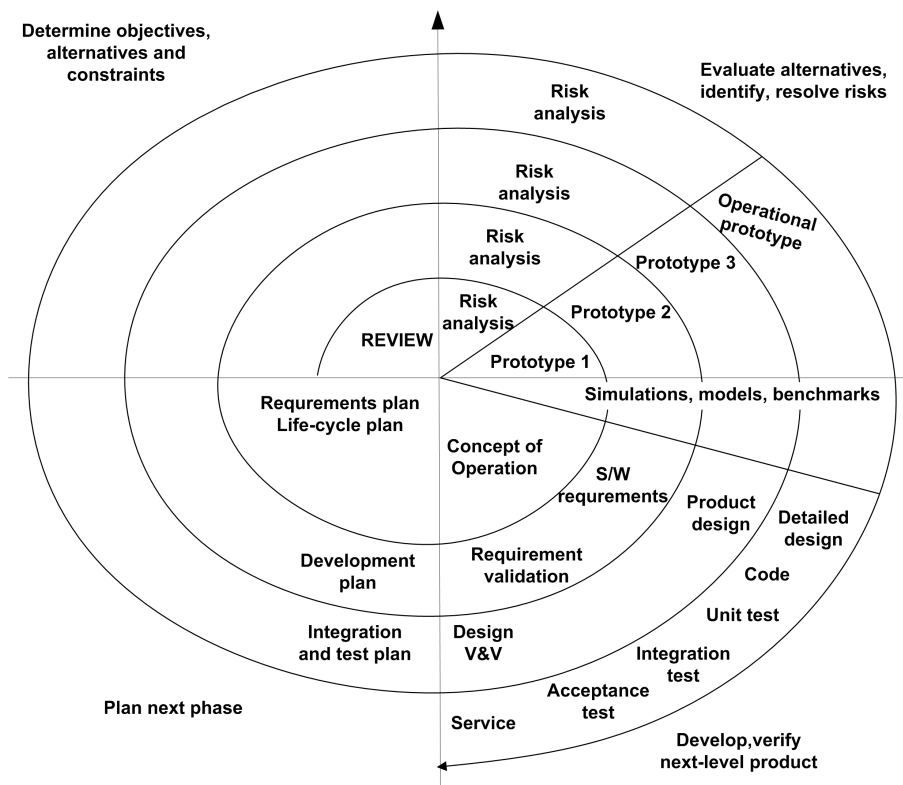


Figure 2.5: Spiral development.

- **Development and validation** - when the software risks are evaluated, appropriate development model is chosen. For example, if the main risks are connected with possibility of poor understanding of the user requirements, best development model may be throwaway prototyping model.
- **Planning** - in the final stage of the loop, the project is reviewed to decide whether to continue with the next loop of the spiral. If the continuation with the next loop is decided, then plans for the next phase of the project are made.

### The Rational Unified Process

In this place it is worth to mention about the Rational Unified Process which is a great example of modern process model which brings together elements from all of three generic models: The waterfall model, Evolutionary development, Component-based software engineering. In this model main drivers are risks and use-cases. RUP identifies four phases in the software process.

**Inception** This phase is strongly connected with business side of the project. The main goal is to achieve concurrence among all stakeholders(entities connected to the project in any way). The result of this phase should be the estimation of the costs

## THE THEORY BEHIND THE SUBJECT

and risks and decision if the project is possible to do and worth of doing. The milestones could be: define significant use-cases, choose candidate architecture, setup project environment. Essential activities in this phase may be:

1. "Formulating the scope of the project.
2. Planning and preparing a business case.
3. Synthesizing a candidate architecture.
4. Preparing the environment for the project."[\[RKd03\]](#)

**Elaboration** "The goals of the elaboration phase are to develop an understanding of the problem domain, establish an architectural framework for the system, develop the project plan and identify key project risks."[\[Som07\]](#) As the result of this phase, we should get stable base for the next Construction phase, which means detailed architecture description and development plan. Essential activities in this phase may be:

1. "Defining, validating and baselining the architecture.
2. Refining the Vision.
3. Creating and baselining detailed iteration plans for the construction phase.
4. Refining the development case and putting in place the development environment.
5. Refining the architecture and selecting components."[\[RKd03\]](#)

**Construction** Now, it is the time to create working system. "The construction phase is essentially concerned with system design, programming and testing".[\[Som07\]](#) As the result of this phase we should get working software ready to deliver to the customer as well as the documentation connected to the project, needed for the users. Essential activities here are:

1. "Resource management, control and process optimization
2. Complete component development and testing against the defined evaluation criteria
3. Assessment of product releases against acceptance criteria for the vision."[\[RKd03\]](#)

**Transition** This phase is basically connected with last correcting and moving the project from development environment to the real environment of the end users. This phase may be very complex as well as very simple, it depends on the type of the product. In most of the software development models, this phase is ignored but in fact it should be threaten as a very important part of the project, since all in all the software is created for the users so concern about their reaction for the ready product should be taken seriously. Essential activities would be:

## THE THEORY BEHIND THE SUBJECT

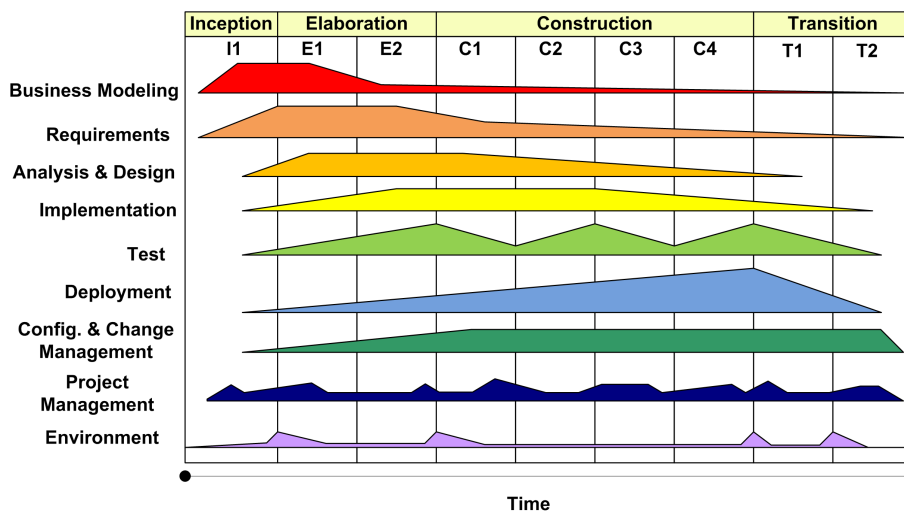


Figure 2.6: The Rational Unified Process.

1. "Executing deployment plans.
2. Finalizing end-user support material.
3. Testing the deliverable product at the development site.
4. Creating a product release.
5. Getting user feedback.
6. Fine-tuning the product based on feedback.
7. Making the product available to end users." [RKd03]

As it can be seen on the figure 2.6, in the RUP there are following workflows:

### "Business Modeling

The business processes are modeled using business use cases.

### Requirements

Actors who interact with the system are identified and use cases are developed to model the system requirements.

### Analysing and Design

A design model is created and documented using architectural models, component model, object models and sequence models

### Implementation

The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.

**Test**

Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementations.

**Deployment**

A product release is created, distributed to users and installed in their workplace.

**Configuration and change management**

This supporting workflow manages changes to the system.

**Project management**

This supporting workflow manages the system development.

**Environment**

This workflow is concerned with making appropriate software tools available to the software development team.”[Som07]

The main activities and models connected with the software engineering has been presented. As we can see, they cover whole software life cycle. The waterfall model is presented as a traditional way of developing software on the other side RUP is an example of the modern approach. In the next chapters we will deal with the knowledge which flow between stages of software engineering processes.

## **2.2 Knowledge management**

We can start the discussion about knowledge management from the definition bellow:

”Knowledge management is a framework and tool set for improving the organization’s knowledge infrastructure, aimed at getting the right knowledge to the right people in the right form at the right time.”[SAA99]

As we can see, this definition is fairly simple in terms of structure. Like in the other areas of the management there is a framework bringing together and ordering a certain activities. Also there is an area of specific tools. If it was the definition of e.g. electricity supply management, then probably because of its transparency, it would not be required to continue the discussion about definition. However, here we have a word *knowledge* as the key object. The meaning of knowledge is probably known by everyone in their own intuitive way, but it is not trivial to explain.

### 2.2.1 Knowledge

It is logical that to talk about knowledge management we must first define what the knowledge is. As it turns out, this question troubled thinkers for a very long time. Already ancient Greece philosophers like Aristotle and Plato were interested in the subject of knowledge. They called it *episteme* - certain and indisputable knowledge, which was opposite to *doxa* - flattering statement. In Greek philosophy there was also a term used to describe knowledge that is strictly practical - *techne*. Work in exploring the intricacies of knowledge continued over the centuries. As a result, there is a section of philosophy called *epistemology*, and the details of it are written in hundreds of books.

This demonstrates the fact that knowledge is not easily definable object. Perhaps for this reason, many authors of literature related to knowledge management often defines knowledge not explicitly. Instead they are using phrases that describe it.

”Knowledge is not what you know but is what you do.”[SAA99]

This quote can be interpreted as, an indication that a true knowledge is a knowledge used in action. Seems to have a profound sense because of the complexity of the real world. Theories developed even by the best theorists still tend to be roughly verified by a reality. This state of affairs succinctly describes the following quotation.

”The illusion of accuracy can be created if people avoid comparison. . . , but in dynamic, competitive, changing environment, illusions of accuracy are short-lived, and they fall apart without warning. Reliance on a single, uncontradicted data source can give people feeling of omniscience, but because those data are flawed in unrecognized ways, they lead to nonadaptive action.”[Wei85]

To define something well, we go to the origin of that object many times.

”Knowledge derives from minds at work.”[DP00]

At present, only the people’s minds are able to create knowledge. Science is still far from clear understanding of what really happens in human head. For example, if we compare a well trained human with the best computer program in the ultimate complex game of Go<sup>2</sup> a human will win. On the other side, even the simplest calculator is better than a human in multiplication. Perhaps the problem is that currently even the best computers, are based on the Cartesian logic and the human mind is probably not.

Of course, there are working definitions of what knowledge is and what distinguishes it from similar entities. When we look at the figure 2.7, we will see the link between data, information, knowledge and wisdom.

---

<sup>2</sup><http://en.wikipedia.org/wiki/Go>

## THE THEORY BEHIND THE SUBJECT

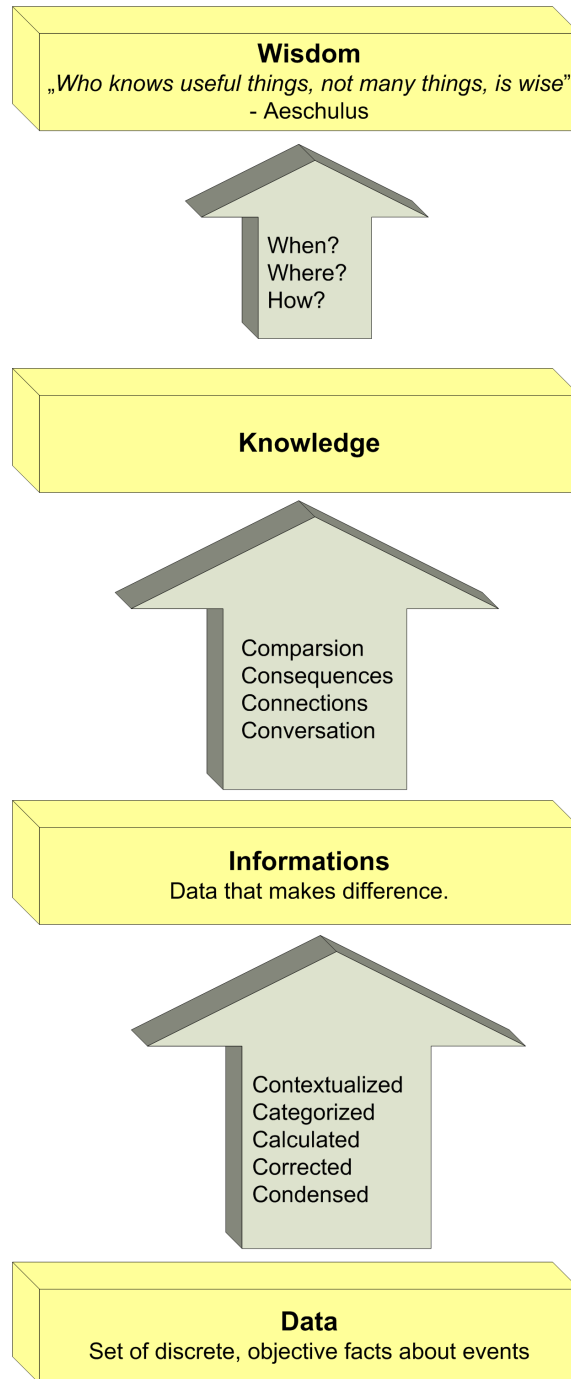


Figure 2.7: Relation of data, information, knowledge and wisdom.

## THE THEORY BEHIND THE SUBJECT

Starting from the bottom, the data is a "Set of discrete, objective facts about events".[DP00] An example of a data set is a telephone billing founded on a staircase. It contains a list of calls made and received by a certain number within a certain period of time. This finding may not be relevant for us, the bare data is quite useless. How Peter Drucker said: "Information is data endowed with relevance and purpose." As can be seen in the diagram, it happens through a set of operations. If we enter billing owners number into our own phone, with little luck, it may belong to our friend. In this case, the data is embedded with a broader context, which may give us a cause for further analysis. When we divide all rows from billing for incoming and outgoing calls we make the categorization of the data. Then we can do some calculations e.g. add the cost of all outgoing calls. Because billing has been laying on the ground for some time, some data has become illegible, we can correct our data by peeling off the damaged lines. The data could also be condensed e.g. by selecting only calls lasting longer than 10 seconds. In this way, the data is transformed into information. Then, by comparing new information with the currently held we can conclude that our friend is talking on the phone much more than we expected, resulting in a very high phone bill. Combining this with our knowledge of the rates of different operators we can come to the conclusion that our friend should necessarily change the provider of telephone services. In this way, the bare data turned into knowledge.

This simplifies description in some way, reflects the process of transforming data into knowledge. Our considerations can be summarized with working definition of knowledge proposed by Prusac and Davenport.

"Knowledge is a fluid mix of framed experience, values, contextual informations, and expert insight that provides a framework for evaluating and incorporating new experiences and informations. It originates and is applied in the minds of knowers. In organizations, it often becomes embedded not only in documents or repositories but also in organizational routines, processes, practices, and norms."[DP00]

Somehow added to the chart on the top appears wisdom, which is probably even harder to define than knowledge. You could identify it as a useful knowledge and skills proper to use in the right place, time and in the appropriate way.

### 2.2.2 Tacit and Explicit Knowledge

At one European Go Congress, a teacher from Japan has told a story. When he first came to Europe to teach Go, he was very surprised by the quantity of questions about the motivation of each sequence of moves he showed. Audience was still asking: "why here?", "why it is more important than that?", "why this is a good move? " etc. At the beginning, he had the impression that players in Europe want to have a specific answer



## THE THEORY BEHIND THE SUBJECT

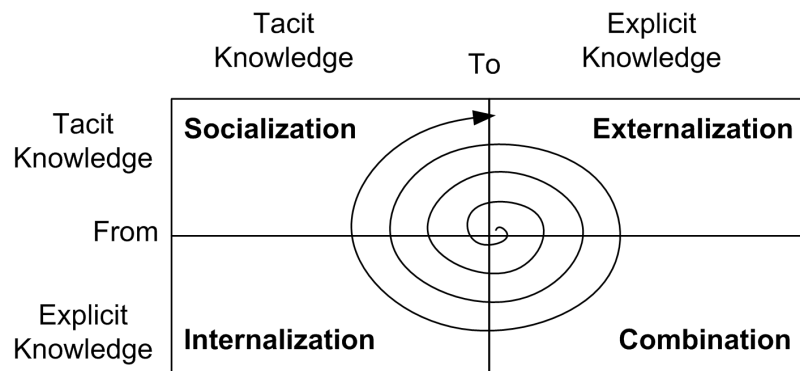


Figure 2.8: Tacit and explicit knowledge[NT95].

for everything, even if it was impossible to give such an answer. Over the years of work in Japan, sensei was accustomed to something almost opposite. There was enough that professional Go player said: "This is a good move." In one book there is an exercise consisting of setting day by day the same situation on the board and repeat: "White has a better position." until we are totally convinced that it is true.

In the culture of the West emphasis is placed on the explicit knowledge. We often do not realize the importance of tacit knowledge.

"Knowledge of experience tends to be tacit, physical, and subjective, while knowledge of rationality tends to be explicit, metaphysical, and objective."[NT95]

Ikujiro Nonaka and Hiroaka Takeushi in their book "The Knowledge-Creating Company" introduced distinction between tacit and explicit knowledge presented in the figure 2.8. The core of their theory is the types of knowledge conversion spiral connecting each field of the diagram. In this way, there are four areas of knowledge conversion:

**Socialization:** From Tacit to Tacit

"Socialization is a process of sharing experiences and thereby creating tacit knowledge such as shared mental models and technical skills."[NT95]

The good example of the socialization process is when a new born baby starts its life. It does not know any language and did not ever go to any school. Nevertheless, it learns rapidly how to move, walk, pronounce the words and talk. It is done only through the observation and imitation of the environment. After this example, it is impossible to deny the enormous power of knowledge achieved in this way.

**Externalization:** From Tacit to Explicit

"Externalization is a process of articulating tacit knowledge into explicit concepts. It is a quintessential knowledge-creation process in which tacit

## THE THEORY BEHIND THE SUBJECT

knowledge becomes explicit, taking the shapes of metaphors, analogies, concepts, hypotheses or models.”[NT95]

Nice example of externalization comes from my friend. She once said in her work that she would like to prepare a pizza. Then her colleague response that he knows how to make it, so she asked him about the recipe. Colleague thought for a moment and then began to dictate - ”Necessary components are the oil, flour and yeast. You must knead them together and put in the fridge for half an hour, to yeast grown. Later, just spread it on to the oiled form and arrange the components you like.” My friend wrote down the whole formula, and ask whether she understood everything correctly. Next day she prepared the pizza precisely with the the given recipe. The pizza was impossible to eat. She went back to her colleague and described what she has done step by step. A colleague was very amused that she did not use the water to make the yeast grow. This ingredient was so obvious to him that he completely forgot about it.

Presentation of tacit knowledge in the form of perfectly corresponding explicit concepts may be difficult. Therefore, in this process there the metaphors, analogies, stories, etc. are often used, which indirectly show the subject of knowledge. It is much easier for human to understand and remember a good story, than a ”cold” logical reasoning.

### **Combination:** From Explicit to Explicit

I believe that academical work on a thesis is a great example of combination. Student is going through many documents about the given subject. The documents contain externalized knowledge from many sources and points of view. Student is combining it and during this process he creates new explicit knowledge.

### **Internalization:** From Explicit To Tacit

”Internalization is a process of embodying explicit knowledge into tacit knowledge. It is closely related to ”learning by doing””[NT95]

When we start to learn how to drive, we have almost no tacit knowledge about it. So we go to driving course where instructor tells us a lot of things. What is a gearbox, when to turn on a lights, where is a break, what a road signs mean etc. In the middle of the course, we are driving with a lot of things in our mind, but we are driving. It can be said that it is ”explicit knowledge in action”. After one year of experience we can easily drive, talk and think about our business simultaneously. Now our driving depends fully on a tacit knowledge.

A spiral in the figure represents a continuous process of passing knowledge through various stages, resulting in a continuous knowledge growing. It is nicely described by quote bellow.

”Knowledge is the only resource that increases with use.”[PRR00]

### 2.2.3 Knowledge management Life Cycle

The figure 2.9 presents knowledge management Life Cycle. It is a good backbone to segregate main knowledge management activities.

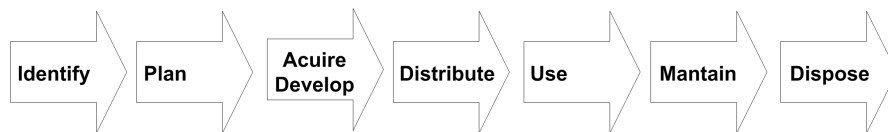


Figure 2.9: Knowledge management Life Cycle[SAA99]

**Identify** In the beginning of any knowledge initiative, we have to identify already existing knowledge. For the purpose of illustration we can create fictional company case.

A manager of infrastructure department in a bank has 100 experienced developers in the COBOL and the IBM Mainframe applications designed for 3270 terminal. The technology has already over 40 years, so it slows down the development process.

**Plan** He wants to implement new technology to improve speed and quality of development process. As current system based on IBM Mainframe and COBOL is very stable and fast, he wants to avoid changing it. Instead, he wants to take advantage of full potential of desktop computers nowadays as tool for developers.

**Acquire and Develop** He puts the experts to find out, if there are any solutions on the market that meets the requirements. Then, he makes contact with software company and set up pilot program. During the pilot program few of employees go through new tool training program and acquire new knowledge.

**Distribute** The pilot program is finished, and he is happy of results achieved by the pilot group. Now, he wants to propagate a new knowledge towards company. There are many principles telling how to do it best, there are also many software solutions which will be reviewed in the next chapter. To keep things simple for usage of this ”Knowledge management Life Cycle” description, lets describe only one proverb.

”The knowledge exchange mechanism:

Knowledge sharing = communication + knowledge recreation.”[SAA99]

We, people, are very diverse. Each of us process the incoming informations in his own unique way. It is that way because of gens, life experience etc. That is why, it is impossible to just "copy knowledge from brain to brain". We learn by combining, what we already know with new information. Our knowledge is more created than copied. That is why, the best way to distribute knowledge is by improving right side of above equation. There are, nowadays, thousands of ways how to improve communication. One of them is by implementing well suited computer system, but we should be aware that it is not enough.

"Do not expect software to solve your knowledge problem." [DP00]

Even the best knowledge management computer system will fail, if people will not use it. Considerations about "how to make people share the knowledge" and "how to improve the knowledge recreation" are beyond of the scope of this document.

**Use the knowledge** Our employees now are using new software to develop and maintain old school applications. They are happier and more effective.

**Maintain** In the active working environment, there is a possibility of arising of problems and bugs. That is why, we should maintain the quality of our knowledge.

**Dispose** When there will be no more COBOL systems left, knowledge about it will be probably no more needed. That is the moment, when the knowledge management chain ends.

Fictional case described above is intended to show general idea of the knowledge management life cycle. In fact there are many conceptions and techniques which can be used in the particular steps. Some of them will be presented in the next sub-chapter.

### 2.3 Software knowledge management

There is a lot of knowledge involved in software engineering processes. Software projects need collaboration of many peoples at all levels of software creation process. Most of the people involved have wide knowledge about computer technology, programming languages, software engineering methods, software testing. Virtually everyone has to know at least something about each software engineering activities.

There are several reasons why software engineering may want to interest in knowledge management. First of all, there is a need to continuous adaptation, because computer world is changing so fast. "Knowledge helps a software organization to react faster and better." [Sch09] Well managed knowledge may help to avoid making the same mistakes. On the other side, it can help recreating good practices and solutions. Both can be done on

## THE THEORY BEHIND THE SUBJECT

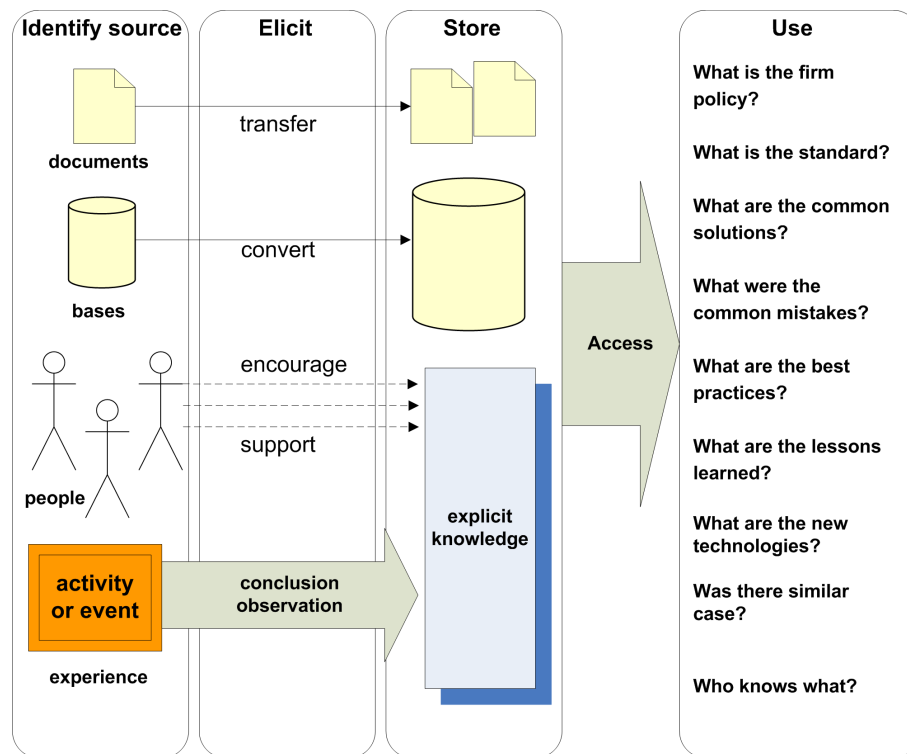


Figure 2.10: General map of knowledge management tasks.

the level of organization, it is called organizational learning. Figure 2.10 presents general overview of knowledge management tasks in software organization.

”We tend to think of learning as a process by which individuals gain new knowledge and insights and thereby modify their behaviour and actions. Similarly, organizational learning entails new insights and modified behaviour. But it differs from individual learning in several respects. Firstly, organizational learning occurs through shared insights, knowledge and mental models. Thus, organizations can learn only as fast as the slowest link learns. Change is blocked unless all of the major decisions makers learn together, come to share beliefs and goals and are committed to take actions necessary for a change. Secondly, learning builds on the past knowledge and experience - which is in memory. Organizational memory depends on institutional mechanisms(e.g., policies, strategies, and explicit models) used to retain knowledge.”[Sta94]

There is also child concept called learning software organization. ”In this context, Feldmann and Althoff have defined a ”learning software organization” as an organization that has to ”create a culture that promotes continuous learning and fosters the exchange of experience”. [BD08]

Rus enumerates several knowledge needs of software engineering company. [RL02]

**Acquiring** knowledge about new technologies. This kind of knowledge is hard to estimate, but quick acquiring and mastering this kind of knowledge is crucial to software organization.

**Accessing** knowledge about domain for which software is being developed. Software development for e.g. chemical company differs from software development for traveling company.

**Sharing** knowledge about local policies and practices. As it was told before, each organization have own polices, practices etc., which can be called in general culture of organization. This knowledge in many cases is shared only at informal meetings. It takes time for new employee to get it in that way.

**Capturing** the explicit knowledge and knowledge "who knows what". Capturing the explicit knowledge will be discussed later. In the software organizations there are naturally emerged experts, who hold knowledge about the specific areas. There are cases, where ability of fast contacting them can be crucial. Unfortunately, they are often hard to find, also, some day they can leave an organization leaving knowledge gap behind. It is possible, that the gap will be noticed only when the knowledge will be needed.

**Collaborating** Software development is the group activity, independent of space and time. Developers can work in different places, like office and home. It is important to keep ability of high level cooperation between members. It means, easy access to a virtual "workplace" and synchronization mechanisms.

### 2.3.1 Knowledge management strategies and generations

Knowledge management concept emerged in mid- 1980s and grows rapidly from the 1990s.[[RL02](#)] First workshop on "learning software organizations" was organized in 1999. [[BD08](#)] Over 20 years of active presence in the business and academic fields of knowledge management created many ideas and tools. There are attempts to categorise it. For purpose of this document only two will be described briefly: main strategies and generations.

As the field becomes more mature it changes and evolves. there are attempts to divide knowledge management into two generations. "In the introduction to the book Challenges and Issues in knowledge management, in the field of management consulting, Buono and Poulfelt claim that the field is moving from first to second generation knowledge management.

**In the first generation** knowledge management, knowledge was considered as a possession, something that could be captured, thus knowledge management was largely a

## THE THEORY BEHIND THE SUBJECT

technical issue on how to capture and spread the knowledge through tools like management information systems, data repositories and mechanistic support structures.

**The second generation** of knowledge management is characterized by knowing-in-action. Knowledge is thought of as a socially embedded phenomenon, and solutions have to consider complex human systems, communities of practice, knowledge zones, and organic support structures. The change in knowledge management initiatives is seen to go from a planned change approach to a more guided changing approach.”[BD08]

According to Hanssen there are two main strategies for knowledge management - codification and personalization.[HNT05] In the area of software engineering knowledge management both are used.

**Codification** can be described as ”people to documents”. In Nonaka model it is connected to process of externalization of knowledge. The main idea is to build best possible computer system to store and receive knowledge. It supports reuse of knowledge, but on the other hand relies on investment in knowledge storing system. In this strategy people should be encouraged and rewarded for using and contributing in the knowledge storing system.[HNT05] It is more related to explicit knowledge and thus to traditional development methods such as the waterfall process.[NB07]

**Personalization** can be described as ”person to person”. In Nonaka model it is connected to process of socialisation of knowledge. The main idea is to develop best possible networks to connect people so tacit knowledge can be shared. In this strategy people should be rewarded for directly sharing knowledge with others. It relies on individual expertise, question ”who knows what” fits in this strategy.[HNT05] It is related more to tacit knowledge and will support agile software development.[NB07] It can be described with quote bellow:

”Knowledge management is about facilitating knowledge sharing by people. It is about increasing their connectivity.”[SAA99]

### 2.3.2 Post-Mortem Analysis

PMA(Post-Mortem Analysis) also called project retrospective is an example of knowledge management tool often used in software engineering from corporations like Microsoft and Apple Computer to medium and small companies.[AdOdSBD07] The PMA is a tool to externalize knowledge and it fits into codification strategy. Maybe it sounds bit sophisticated, but main concept behind it is very simple. It can be described in two steps:

1. Gather all participants from just finished project

2. Ask them to identify the aspects of project which:
  - (a) Worked well and should be repeated in the future.
  - (b) Worked badly and should be avoided in the future.
  - (c) What was merely "OK", but leave room for improvement.

[SDHM03]

As the main idea behind PMA is so simple, the tool is flexible and can be used in many ways. It is flexible in:

- **Phase of the project** - it can be done after finishing whole project or after reaching milestone
- **Context** - questions can be asked from general to very specific aspects like programming techniques used
- **Time and place** - it can be done as short(e.g. 2 hours with all + 1 hour with team leader) interview with all participants at once or on the individual meetings
- **Structure** - it can be relatively informal or formal and precisely structured

[AdOdSBD07]

PMA helps to avoid "lost knowledge" problem. As Anquetil claims, it is a huge problem in a software maintenance. It takes from 40% to 60% of software maintenance effort to understand the system maintained. "The knowledge is "extracted" at great cost from detailed analysis of the system's source code." [AdOdSBD07] On the other hand, it is relatively easy to capture this knowledge during a software creation.

## 2.4 Tools for software knowledge management

As the software engineering is about effectively creating the computer software, there are many applications to support SE activities. We are talking about tools for knowledge management in software engineering, so about connection of two disciplines. We can approach it from direction of software engineering tools and knowledge management tools. We begin from software engineering side. Briefly describing in the order of the waterfall model we have:

**Software specification** is theoretically possible in the simplest case by meeting the client once and trying to imagine and remember "what he wants?". In the real life it takes many documents, UML(Universal Modelling Language) diagrams and discussions of various kind to create professional specification. The software to support this is focused on improvement of client-engineer communication, quality of exchanged



documents and diagrams. We can name Microsoft Visio as the leading diagrams creating tool.

**Software design** also in the simplest case is possible just in the mind of the engineer. To support design of bigger projects there are many tools mostly based on UML. They can even create source code from UML diagrams. We can name here open source ArgoUML<sup>3</sup>.

**Software implementation** is also possible by using only the simplest text editors like "notepad" to write a source code. But everyone, who actually tries it knows that it is hard. That leads to extending text editors with various features to make software development easier and more efficient. Source code's syntax highlighting to avoid syntactic mistakes and make the code more "visible" for humans is one of the basic improvements. Later, it becomes possible to fold and unfold parts of code, editor starts to make suggestions about keywords, variables etc. When text editor starts to organise whole programming project, enable to create GUI by moving the building blocks by the mouse and debugging whole application it cannot be called "text editor" anymore. It becomes IDE(Integrated Development Environment). Leading open source projects are Eclipse<sup>4</sup> and NetBeans<sup>5</sup>, a well-known commercial solution is Microsoft Visual Studio.

Because the source code changes during development and in most cases there is more than one developer, revision control systems were created. Very popular examples are SVN and GIT.

**Software verification and validation** is important because of complexity of computer software. It is almost impossible to create software without bugs. That is why there are bugs tracking systems like Bugzilla<sup>6</sup> or Mantis Bug Tracker<sup>7</sup>.

The applications described can be somehow called knowledge management tools as they support codification(e.g. in form of diagrams) and personalization(e.g. by the UML diagrams to support engineer-client communication). But it is only on the level of one project. Previously, we have talked about management of the whole software engineering institution knowledge. To accomplish this task different software is needed.

Microsoft Team Foundation Server is one of the complete solutions for managing whole life-cycle of team projects. It contains work item management, source control, bug tracking, reports generator and detailed guides for Microsoft versions of software development processes. It uses Windows SharePoint Services as a web portal and document

---

<sup>3</sup><http://argouml.tigris.org>

<sup>4</sup><http://www.eclipse.org>

<sup>5</sup><http://www.netbeans.org>

<sup>6</sup><http://www.bugzilla.org>

<sup>7</sup><http://www.mantisbt.org>

repository which is integrated with Microsoft Office Applications.[[Mic10](#)] Along with the Visual Studio and Microsoft Windows of course, it virtually covers all needs of software engineering process. On the other hand, it can be an expensive solution and the users are forced to learn and follow Microsoft's way of doing things.

The main rival on the market for the Microsoft's solution is Jazz from IBM. "Jazz is an IBM Rational initiative to help make software delivery teams more effective."[[IBM10](#)] It consists whole family of "Rational" products. The main one is Rational Team Concert with similar features to Microsoft TFS. It helps in management of work items, has source control, building the management, iteration planning support and automated reports generation. To collaboration on the documents it can use Lotus Quickr and Microsoft SharePoint. To support social networks it can connect with Lotus Connections or Microsoft Office SharePoint Server's social networking features. The main idea again is that a Team is the most important in the software development. The second important thing is that, as IBM says they are intended to "break the walls between the tools". That is why IBM solution can or have to collaborate with software from various vendors. Of course, this solution is also commercial and can cost a lot.

On the other side of the commercial wall, there are open source solutions. We can use Redmine<sup>8</sup> for the project management, along with GIT or SVN for source control, NetBeans or Eclipse as the main IDE, wikis as the main documentation holder, XMPP(Extensible Messaging and Presence Protocol) for communication, Linux as operating system etc. We can also build social network with e.g. Elgg<sup>9</sup>. In fact, there is many possibilities in the open source world to build functionality in many aspects similar to provided by commercial tools described above. It all comes free of charge and can be precisely suited to needs of institution. It sounds great, but there is always second side of a coin. In the case of open source applications we have responsibility for connecting all the parts together and maintaining it. As the most popular open source licence GNU GPL(General Public License) says:

"This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details."[[Fre07](#)]

In the case of open source solutions, institution does not have to pay for a software, but does have to have open source solutions specialists. To find the best suited solution, configure and maintain it on the level of organisation certain knowledge is necessary. In the case of commercial solutions a lot of job is already done, but on the other hand, it is expensive and we have to do things in the way proposed by the solution seller.

---

<sup>8</sup><http://www.redmine.org>

<sup>9</sup><http://elgg.org>

## THE THEORY BEHIND THE SUBJECT

Software engineering side of tools has been briefly described. Now, we will jump to the opposite point of view. It can be summarized by the question: "What tools used for knowledge management in general can be useful for software engineering?".

Now, we will take a closer look on the already mentioned solutions - Microsoft Share Point and IBM Lotus® Quickr®.

Share point is a collaboration portal allowing to share documents and other information through web based portal between end users in the company. The name "Share Point" refers to several products, the most important are WSS(Windows Share Point Services) and MOSS(Microsoft Office Share Point Server). WSS is a free of charge addition to Windows Server and can be downloaded from Microsoft webpage. It provides many features including CMS, blogs, wikis, documents library, bug tracking, search engine and many more. It is mainly foundation to the MOSS, but well used can be very useful for small and medium companies. For the enterprise clients Microsoft offers MOSS. It is a commercial product, based on a WSS core.

IBM's product in the same area is the IBM Lotus® Quickr®. It contains similar features like wikis, Blogs, File Sharing, Documents Libraries, Forums etc.

### 2.4.1 Wikis

In 2010 probably majority of the Internet users know at least one wiki - Wikipedia presented in the figure 2.11. A story of wiki started in 1994, when Ward Cunningham began to develop WikiWikiWeb - the first wiki. He originally described it as "the simplest online database that could possibly work". At the beginning of his work, he created number of wiki design principles:

- **Simple** - Easier to use than abuse. A wiki that reinvents HTML markup ([b]bold[/b], for example) has lost the path!
- **Open** - Should a page be found to be incomplete or poorly organized, any reader can edit it as they see fit.
- **Incremental** - Pages can cite other pages, including pages that have not been written yet.
- **Organic** - The structure and text content of the site are open to editing and evolution.
- **Mundane** - A small number of (irregular) text conventions will provide access to the most useful page markup.
- **Universal** - The mechanisms of editing and organizing are the same as those of writing, so that any writer is automatically an editor and organizer.



Figure 2.11: Wiki Wikipedia page.

- **Overt** - The formatted (and printed) output will suggest the input required to reproduce it.
- **Unified** - Page names will be drawn from a flat space so that no additional context is required to interpret them.
- **Precise** - Pages will be titled with sufficient precision to avoid most name clashes, typically by forming noun phrases.
- **Tolerant** - Interpretable (even if undesirable) behavior is preferred to error messages.
- **Observable** - Activity within the site can be watched and reviewed by any other visitor to the site.

- **Convergent** - Duplication can be discouraged or removed by finding and citing similar or related content.”[Cun94]

As time shows this principles are brilliant. Wikis rapidly spread around the world. There is number of wiki implementations, both commercial and opensources. One of the most popular is ”MediaWiki” originally for use on Wikipedia<sup>10</sup>. Implementations differ in details and purposes, we will describe few for a general view of the subject:

**TiddlyWiki** <sup>11</sup> is nice to begin with, because it is the smallest possible one. In fact, it is only one file! It is simple and useful, for small projects like personal notes. It can be easily send by the email or held on pendrive.

**doxWiki** <sup>12</sup> is a bigger one, but still very small(about 200kB after installation) wiki implementation. It comes with simple HTTP server written in perl so there is no need to set up ”big” web server like Apache. It needs only perl and web browser, so it also can be easily take away with an ”USB thumb”.

**dokuWiki** <sup>13</sup> is a wiki implementation suited for small and medium company’s documentation needs. It is written in PHP and uses plain text files to store pages. It has already about 500 plugins.

**Twiki** <sup>14</sup> is aimed to enterprise market. ”It is typically used to run a project development space, a document management system, a knowledge base, or any other groupware tool, on an intranet, extranet or the Internet”.

**TikiWiki** <sup>15</sup> is all in one solution. It brings along wiki, forums, blogs, bug tracker etc.

Wikis, because of a great conceptions behind, are very useful as data repositories and collaboration portals. As it was briefly presented before, commercial solutions from two main vendors are actually based on existing Web 2.0 ideas interconnected with existing desktop software. One could argue - if it is really necessary to buy such software, if we can use just wiki instead, in which there are parts of this software anyway. In considering purchase of tool like this, is important to ask ”what value is added by this tool and if it is not the well known wine in the fancy new bottle”. Flexibility and scalability are the main features of wiki which allows to appear as a single project’s page, but also as a corporate portal. The usage depends only on the vision of the user and is not limited by ”hard coded” parts. In the wiki everything is a page, and creating a page is very easy. Each page

---

<sup>10</sup><http://www.mediawiki.org>

<sup>11</sup><http://www.tiddlywiki.com>

<sup>12</sup><http://doxwiki.sourceforge.net>

<sup>13</sup><http://www.dokuwiki.org>

<sup>14</sup><http://twiki.org>

<sup>15</sup><http://tikiwiki.org>

can contain various content like plain text, formatted text, pictures, movies, diagrams, LaTeX equations, highlighted source code and attachments of any kind. Still, because of a unified style, pages remain easily readable. Each wiki page is a plain text, which makes it easy to handle with external tools. A variety of plugins extends the flexibility of wikis even further.

### 2.4.2 Semantic wikis

Semantic wikis follows the idea of implementing current Semantic Web technologies into wikis. In this way wiki content can become "computer readable". We begin our trip to semantic wikis by fast overview of core semantic web technologies. Then, the discussion about semantic wikis on the example of Semantic MediaWiki<sup>16</sup> will be taken.

Semantic Web is an idea of Tim Berners-Lee who created WWW standard and the first Internet browser. The difference to already existing standards is that in Semantic Web data have to be "understand" not only by humans, but also by computers. It can be done by describing all data by unified metadata. Later, it will be possible for computers to do several operations on the described data like: improved searching, combining and reasoning. Semantic Web is based on existing, well known standards like HTTP extended by new ones on different abstraction level.

"Semantic Web it is collection of standard technologies to realize a Web of Data." [Her09]

To grasp the concept of Web of Data we first can look at the current situation. For example, there are few social networking portals like Facebook or LinkedIn. If we fill in our profile data and add our friends at one portal and we want to use another one we have to do the whole work again. This is because there is no connection between those two applications which held the personal data. It can be called "Web of Applications". In the opposite, in "Web of data" data chunks or generally speaking resources can be interchangeable between applications. It can reduce redundancy, revision problems and can have huge impact on the Internet as whole. The collection of basic ideas to implement Semantic Web is described bellow.

**URI(Uniform Resource Identifier)** is a standard used to resource identification. In particular, URI is used to identify network resource. It is a parent concept to the URL(Uniform Resource Locator)<sup>17</sup>.

The examples of URI are:

- <http://www.server.com>

---

<sup>16</sup><http://semantic-mediawiki.org>

<sup>17</sup><http://tools.ietf.org/html/rfc2396>

- ftp://ftp.server.com
- mailto: name@server.com

**RDF(Resource Description Framework)** is a language created by W3C<sup>18</sup> to describe Web resources in computer understandable form. It is a solution for enormous amount of all kinds of data like: text, pictures, videos, sound files etc. in the Internet. It is hard to manage or systematize this data. It can be compared to mixed pages of book without table of contents or to box of dvd disks without labels. Table of content in a book is a kind of "metadata", it is data about data. There are already some examples of metadata in the Internet, like XHTML <meta> mark-up. But, unfortunately, they are too ambiguous and not unified. The goal of RDF is to create universal worldwide specification of describing metadata.

RDF can be based on XML, and consist of three main elements, called "triple", to describe data. The example of triple is shown in the figure 2.12. These elements are actually an extended link. It describes two ends of link e.g. "This thesis" and "wikis" and relation between them e.g. "is about".

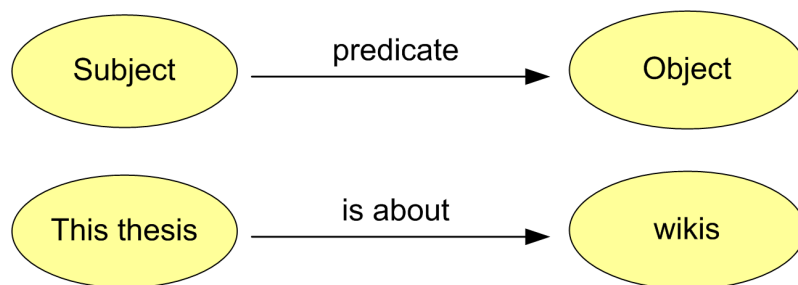


Figure 2.12: RDF triple

**SPARQL(SPARQL Protocol And RDF Query Language)** is a query language created to search through RDF metadata.

These three elements are enough to catch basics of Semantic Web, although there are more specifications and work is in progress. For example, in February 2010 W3C opened Semantic wiki on Semantic Web standards<sup>19</sup>.

There are already several implementations of Semantic wiki like Platypus Wiki<sup>20</sup>, IkeWiki<sup>21</sup> and Semantic MediaWiki<sup>22</sup>. To show the power of semantic wikis several features that distinguish Semantic MediaWiki from MediaWiki on the example of MediaWiki based Wikipedia can be presented.

<sup>18</sup><http://www.w3.org>

<sup>19</sup>[http://www.w3.org/2001/sw/wiki/Main\\_Page](http://www.w3.org/2001/sw/wiki/Main_Page)

<sup>20</sup><http://platypuswiki.sourceforge.net>

<sup>21</sup><http://www.ikewiki.com>

<sup>22</sup>[http://semantic-mediawiki.org/wiki/Semantic\\_MediaWiki](http://semantic-mediawiki.org/wiki/Semantic_MediaWiki)

## THE THEORY BEHIND THE SUBJECT

```
Category:1879 births
Category:1955 deaths
Category:19th-century German people
Category:20th-century German people
Category:People from Ulm
Category:Academics of the Charles University
Category:Albert Einstein|Albert Einstein
Category:German Jews
Category:German Jews who emigrated to the United States to escape Nazism
Category:German-language philosophers
Category:German Nobel laureates
Category:German pacifists
Category:German philosophers
Category:German physicists
Category:German refugees
Category:German socialists
Category:German vegetarians
```

Figure 2.13: Albert Einstein page's categories in Wikipedia.

**Search** in Wikipedia we have only a possibility to text search. It is impossible for e.g. to ask about all board games. If there is a page where words "board game" are mentioned it will be in the result of search, even if the whole sequence will be "it is the opposite to board game". Semantic wiki can be more like query to the relational database.

**Lists** in Wikipedia for e.g. the biggest cities in Europe sorted by the population must be generated manually. But all big European cities have its unique pages in Wikipedia with population included. If population of one city changes, we have at least two places to change it. With semantic wikis it is possible to automatically generate such lists by a proper query.

**Categories** in Wikipedia are widely used for structuring data. In the figure 2.13 are presented just few categories to which Albert Einstein's page belongs. In this way it is possible to find all Nobel laureates by browsing proper category. But categorization structure like this is for sure hard to generate and maintain. In semantic wiki, again, it is possible to achieve similar functionality by proper query.

**Export** to the external applications from Wikipedia can be done in a very limited way. It is possible to export whole Wikipedia to an offline version e.g. for mobile phones<sup>23</sup>, we can also imagine external use of Wikipedia's categories. Other data are mostly

<sup>23</sup><http://www.legaltorrents.com/torrents/526-wikipedia-for-mdict—may-2009>



## THE THEORY BEHIND THE SUBJECT

impossible to export automatically. With semantic wiki situation is different, data with RDF metadata can be easily shared through the network.

## THE THEORY BEHIND THE SUBJECT

## Chapter 3

# WeakType plugin

### 3.1 WeakType idea

Wikis are already very useful tools for software engineering. However, there are still functionalities that can be implemented into wikis to support software engineering even better. It is especially interesting because wikis are in many cases open solutions encouraging people to improve them. Semantic wikis presented before are extending existing wiki conception by the new one. Classic wikis are totally free in the terms of structure and formalization of the data. This sometimes can lead to "informational mess". As it was shown, Semantic Web conception implemented in wikis is highly beneficial in the terms of structuralization of data. Unfortunately, there is also the other side of the coin - Semantic Web technology's ease of use.

"We need to make this technology dramatically easier to use, for both developers and end users. Semantic Web technologies are difficult conceptually, even for experienced software developers. It's difficult to imagine nontechnical end users effectively authoring RDF statements without software assistance. Even for sufficiently trained users, writing the required precise statements takes much more time and effort than writing informal text."[Sou05]

Authors of white paper "Incremental Knowledge Acquisition in Software Development Using a Weakly-Typed Wiki" presents different idea of making wiki data more structured. They introduces Weaki - wiki with weak types[CFFA09]. In the Weaki there are special purpose pages called types. They are intended to provide structure for the other pages. The conception of templates for wiki pages is well known. For example, when we know that in our wiki will be many pages describing Use-Cases, we can create Use-Case template page. This page will have only the structure, no content inside. Later, when we want to create new Use-Case page, we can start from the Use-Case template and fill it in with specific content. Weaki idea is in some way opposite. We start from creating a "normal" page and, after it, content reorganization according to chosen type is possible.

The problem of this practical part of the thesis is to integrate elements of Weaki conception into one of the well known wiki engines. It can be valuable to use in software engineering knowledge management as it will support codification of knowledge.

## 3.2 Creation of the plugin

The problem introduced before is to create wiki with support of weakly typed pages. This chapter is intended to present the creation process of this tool. The final result of development should be the prototype code showing "weakly typed idea" in action. Below are several principles that the tool should follow.

- **Tolerant** - "Interpretable behavior is preferred to error messages." [Cun94]
- **Organic** - The structure and text content of the site are open for editing and evolution.
- **Everything is a page** - Authors may create and edit types through the same mechanism used to create and edit contents, as both types and contents are authored as regular wiki pages. [CFFA09]

### 3.2.1 Specification

We begin with normal wiki page like this one in the figure 3.1.

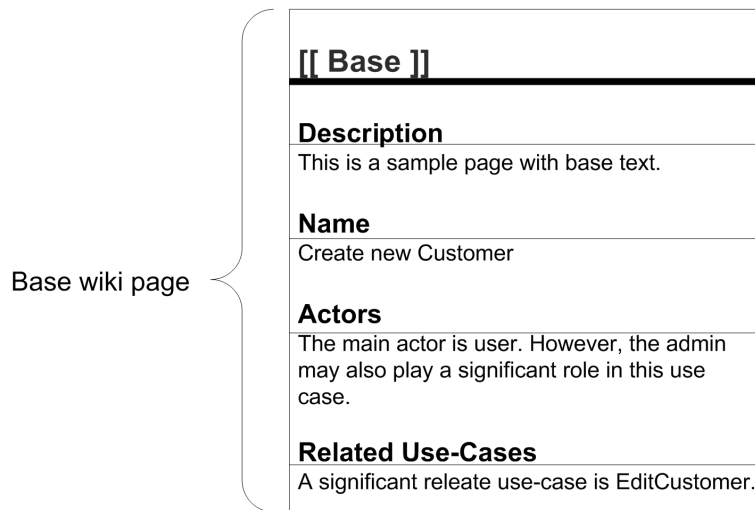


Figure 3.1: Base wiki page.

As we can see, this page contains 4 headers and some text. Of course, in a real case, a page can be much more complicated, but probably almost always it will contain one or more headers to group the data. That is the reason, why the headers are the most

## WEAKTYPE PLUGIN

important for us in this prototype. The tool should collect headers structure and look for similar types. "Type" in Weaki conception is a wiki page with defined structure intended to be a template for other pages. We can see an example of the type page in the figure 3.2. It contains only headers organized in such way, that it defines a "use case" page.

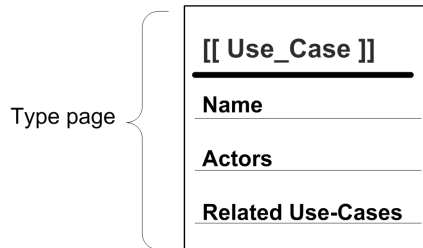


Figure 3.2: Type page.

The tool should collect all headers from the base page and look for types with similar structure. As a result, it should produce table with names of matching types, level of matching and link to the second stage of processing. Tool can be activated by some new wiki markup. Our base page with tool markup should look similar to the figure 3.3.

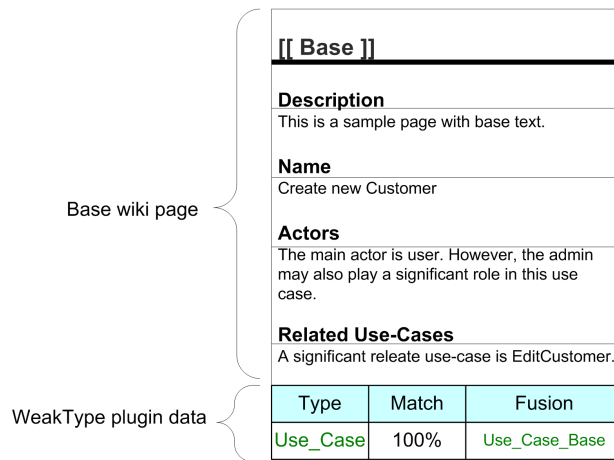


Figure 3.3: Base wiki page with WeakType plugin data.

As we can see, the "Use-Case" type from figure 3.2 has been found and presented in the generated table. Matching rate is 100%, because all headers presented in the type was present in the base page. And finally, there is a link to the new, based on the type, page. The second part of the tool processing begins with mouse click on that link. The Tool should organize all matching headers in the way presented by the type. All the data that does not match should be added below after special mark. As the final result, we should obtain the page similar to the figure 3.4.

Whole process is presented in the figure 3.5.

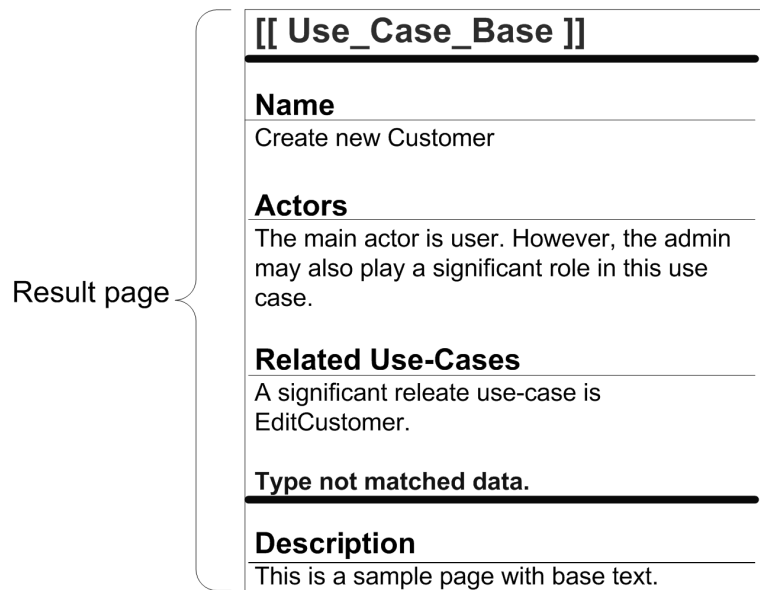


Figure 3.4: Result page.

### 3.2.2 Architectural design

Whole code should be integrated with the well-known, already existing wiki implementation. There are several reasons to support this idea. First of all, as we have seen before, there are many wiki engines already on the market, most of them are available totally free of charge and supplied with full source code. The license allows and even invites to use them as the base for various applications. In many cases they are mature projects with years of continuous tests and improvements. Second reason is that the Weaki conception is based on wiki foundations. All the basic concepts are preserved and only few of them are improved. As the purpose of this tool is only to show the idea in work, it would be a massive waste of effort to build all the platform from scratch.

The tool will be based on the DokuWiki<sup>1</sup>. DokuWiki was created by Andreas Gohr in June 2004. It is currently one of the most popular wiki engines in use<sup>2</sup>. It is released under terms of GNU General Public License and the source code is well-documented. DokuWiki is written in PHP language and based on plain text files. It has a generic plugin interface, which simplifies the process of writing and maintaining plugins. There are over 581 plugins<sup>3</sup> already available. Basing on these facts, it was decided that creating the tool as a DokuWiki plugin is the best choice. As the DokuWiki is free of charge and cross-platform (it basically needs only a working web server), there is no need to pay for operating system neither. Ubuntu Linux<sup>4</sup> together with Apache Web server and PHP are

<sup>1</sup><http://www.dokuwiki.org>

<sup>2</sup><http://www.wikimatrix.org/stats.php>

<sup>3</sup><http://www.dokuwiki.org/plugins>

<sup>4</sup><http://www.ubuntu.com>

# WEAKTYPE PLUGIN

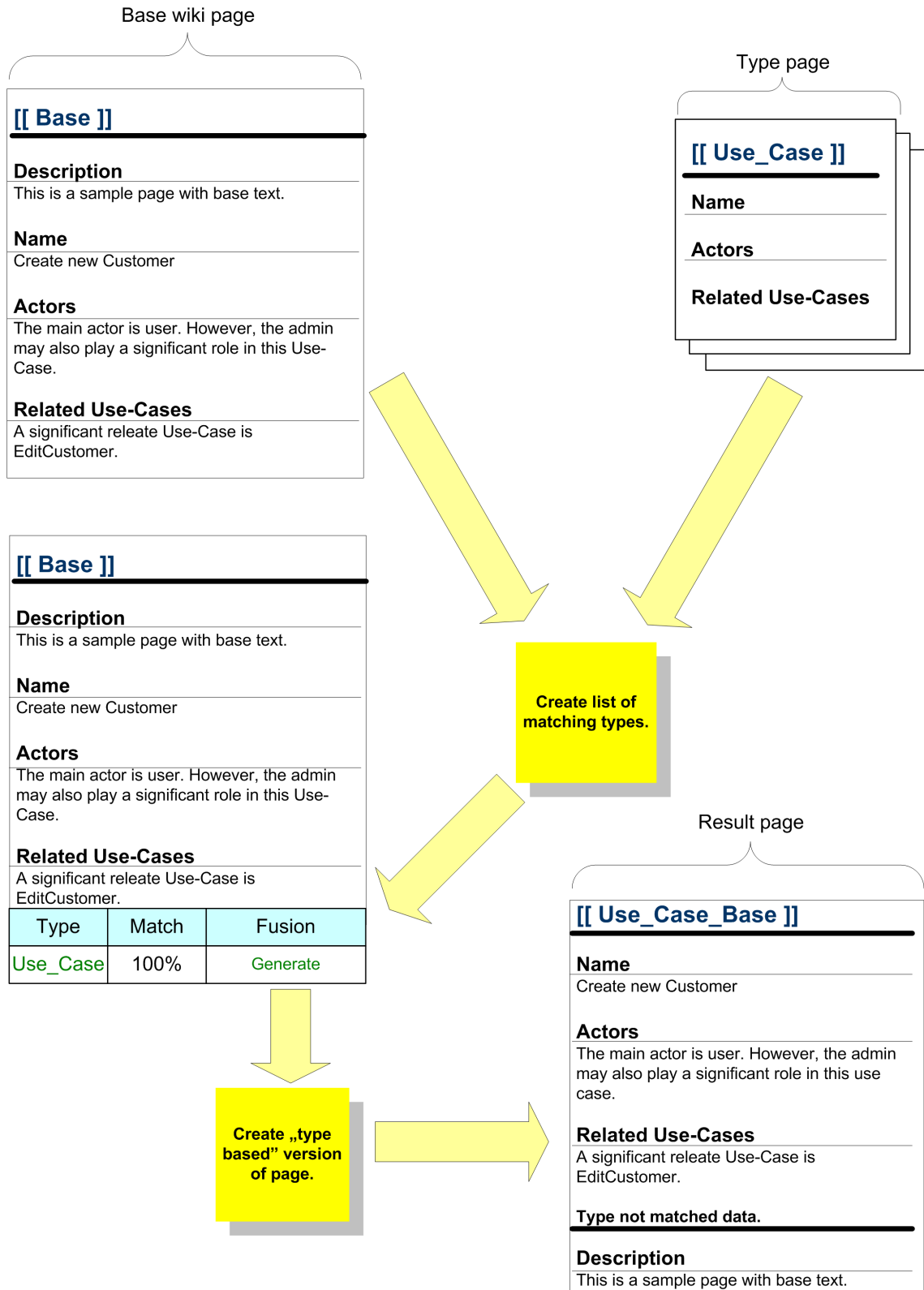


Figure 3.5: General view of whole process.

## WEAKTYPE PLUGIN

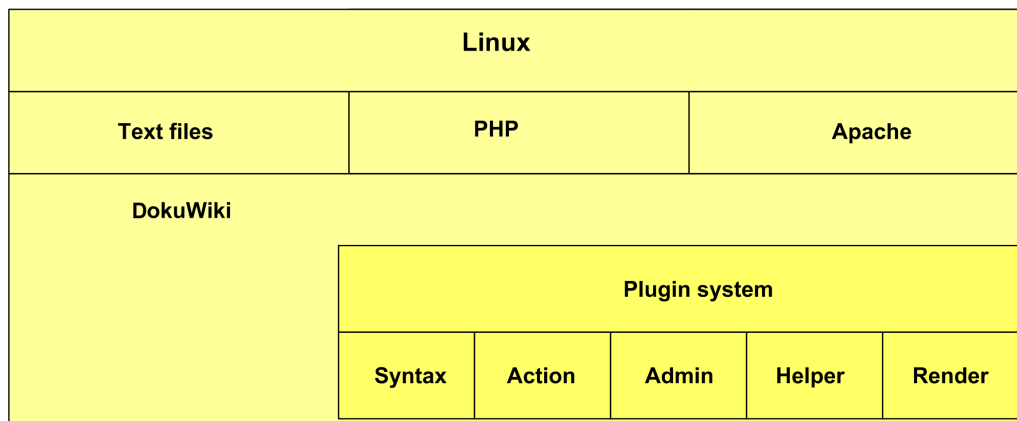


Figure 3.6: Architectural design.

stable base for the tool. Whole architecture is presented in the figure 3.6.

Work with plugin system is good to begin from reviewing the list of already available plugins<sup>5</sup> and reading "Development Resources" page of DokuWiki documentation<sup>6</sup>. "At the moment, DokuWiki features five different plugin types:

**Syntax Plugins** extend DokuWiki's basic syntax.

**Action Plugins** can be used to extend or replace many aspects of DokuWiki's core operations, from saving wikipages to adding new action modes.

**Admin Plugins** can provide administration functionality for DokuWiki - these plugins are accessible to superusers and managers via the Admin button.

**Helper Plugins** can be used to provide functionality to many other plugins, so each plugin does not have to re-implement a certain function over and over again.

**Renderer Plugins** allow to create new export modes and to replace the standard DokuWiki xhtml renderer."[\[dok10\]](#)

### 3.2.3 Interface design

One of the strong side of wikis is that all sites and functionalities are unified in the term of interface. There are all clean(without unnecessary decorations) and thus easy to read and navigate. Interface of the tool should be well integrated into this style.

The tool in the term of interface contains several parts. Relations between them are presented in the figure 3.7.

<sup>5</sup><http://www.dokuwiki.org/plugins>

<sup>6</sup><http://www.dokuwiki.org/development>



## WEAKTYPE PLUGIN

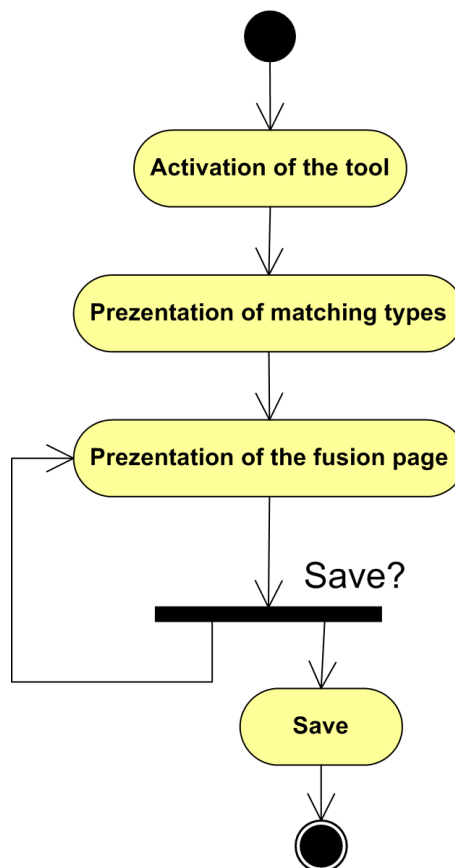


Figure 3.7: Relations between interface elements.

**Activation of the tool** will be done as a new wiki markup - `[WeakType]`. It is the core idea in wikis to use the markups in the pages, so one additional markup fits in general conception.

**Presentation of matching types** will be done as a yellow box at the top of the page. The yellow color is common in DokuWiki to present additional features. It will distinguish data generated by the tool from the content of the page. The box will contain a table presented in a DokuWiki style. The table will contain one row for one found type. The row will be divided into three fields: wiki link to the type page, indicator of "how much of the page match the type" and the link to generate the fusion page.

**Presentation of the fusion page** will be done similarly to the presentation of the matching types to keep consistency of the interface. It will look as a wiki page with yellow box above. The box will contain short explanation and two buttons - save and cancel.

**Not matching data separator** will be created by using standard wiki markup. There will be the text "Not matching data" surrounded by two horizontal lines.

### 3.2.4 Component design

Analise of the DokuWiki plugin system shows, that to reach desired functionality two main components are needed.

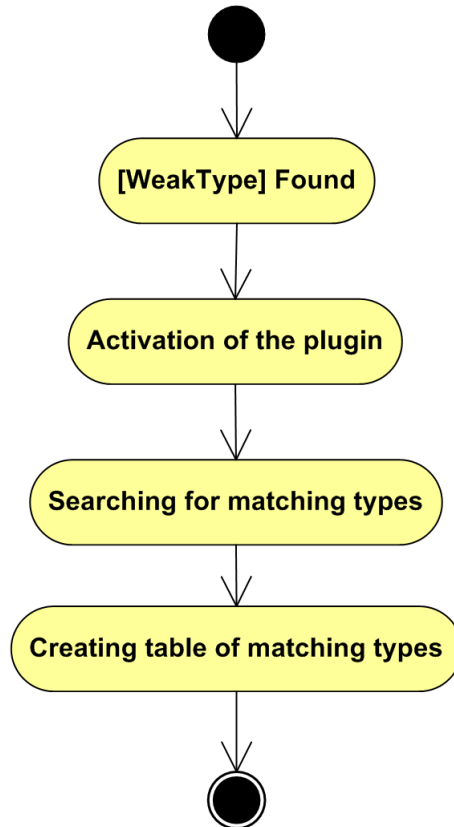


Figure 3.8: States of the syntax component.

**Syntax component** will be implemented as DokuWiki syntax plugin. The goal of this component is to add a new recognized marker to the DokuWiki syntax. After the marker is found this component should compare actual page with all types pages and generate "presentation of matching types" box (figure 3.8).

**Action component** will be implemented as DokuWiki action plugin. It will be activated by the link from the table created by syntax component. After activation, it will combine structure taken from type and content taken from base page to create "presentation of the fusion page". This component will be also responsible for saving result page (figure 3.9).

# WEAKTYPE PLUGIN

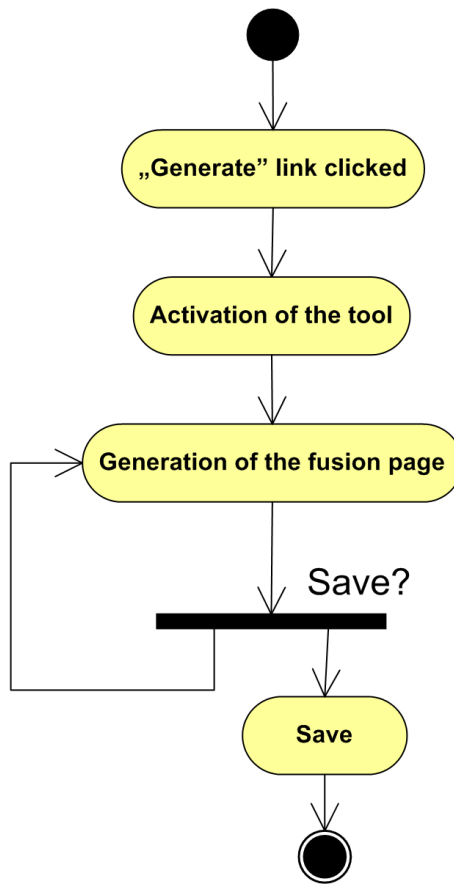


Figure 3.9: States of the action component.

### 3.2.5 Data structure design

The wiki page is divided into sections by headers. The headers can be nested and overall they create tree structure with root in level 0.

```
=====  
===== level1 =====  
===== level2 =====  
===== level3 =====  
===== level4 =====
```

Figure 3.10: Source code of the headers in DokuWiki.

In the DokuWiki headers are represented like in the figure 3.10. For the example, if this chapter will be a DokuWiki page, its source code will looks similar to the figure 3.11. In the next figure 3.12 it is presented as a tree of headers.

```
=====  
===== Chapter 3 =====  
===== WeakType idea =====  
===== Creation of the plugin =====  
===== Specification =====  
===== Architectural design =====  
===== Interface design =====  
===== Component design =====  
===== Data structure design =====  
===== Algorithm design =====  
===== Implementation =====
```

Figure 3.11: Structure of this chapter as a DokuWiki headers.

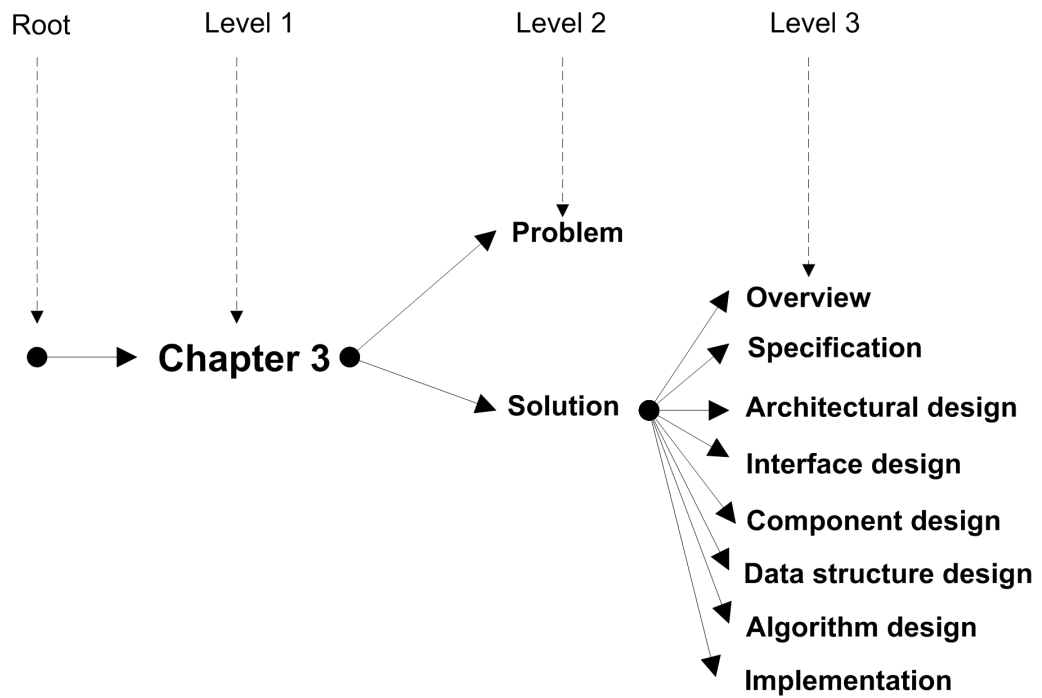


Figure 3.12: Structure of this chapter as tree of headers.

### 3.2.6 Algorithm design

#### Find matching types

The goal of this process is to create a table with matching pages. Each row contains one special wiki page called **type** which is somehow similar to the base page. The tricky part is to find out which type page is similar and how much.

Currently, it is done in the few steps:

1. Get all headers from current page.
2. For each header run fulltext search in the types namespace for text matching pages.
  - (a) For each result add matched type name into output table
  - (b) If there is more than one hit to one type, add 1 into match column(in the picture there is 100%, but in current version it is just a number of hits)

#### Create type based version of the page

This step is to create type based **fusion** of the base page and the type page. It should be done by combining the type structure and the base content.

Currently it is done in two passes:

1. Algorithm presented in the figure [3.13](#) creates arrays containing:
  - (a) all data matching to the type
  - (b) all not matching data
2. Algorithm presented in the figure [3.14](#) recreates matching array in order to:
  - (a) add other type headers
  - (b) preserve type order of headers

## WEAKTYPE PLUGIN

```
***** pass 1 *****
* Get match and not match base lines *
*****
read the input files
get lines from base file into array
get headers from type file into array
add informational line at the beginning of "not match lines" array
for each line of base file
    ?cut off weaktype activation line
    get level of header(if the line is not header the returned
    level is irrelevant)
    if (current line is a header) and (header level equal or
    below previous founded one)
        // we have found header interesting line
        // let's find out if there is matching header in the
        array of type headers
        for each type header from type headers array
            get the text of the header
            get the level of the header
            get the markup of the header(e.g. ==== title ====)
            get the title of base line header (cut off all "=")
            if base line header text it the same like the type
            header text
                // have found matching header!
                put the founded title into Match array
                set found indicator
                // The level is taken from the type so it is
                // type based in this internal iteration there
                // is nothing else to do
            if not found any matching header form type file,
            put the header into NotMatch output array
        else - the line is not header or the header level is higher
        than last founded one
            if found indicator is set
                Put the line into Match array
            else
                Put the line into NotMatch array

// Now, we have a completed NotMatchBaseLines array and
// MatchBaseLines with matching headers and content
// Since the result page should be type based, we need to add
// not matching headers from the type array to preserve
// the order of type page it was best to do the second pass.
```

Figure 3.13: Pass 1: Get match and not match base lines.

## WEAKTYPE PLUGIN

```
***** pass 2 *****
* rebuild MatchBaseLines array, add rest type *
*** headers, preserve type order of headers ***
*****
for each type header
  for each line from Match array
    get level of header(if the line is not header the
    returned level is irrelevant)
    if (current line is a header) and (header level equal
    or below previous founded one)
      // it is a interesting header line
      if it match with header line
        put it into the Match array
        set found indicator
      else - it does not match
        set found indicator false
    else - the current line is not a header or the level
    above previous founded one
      if found indicator is set
        add it to output array
  if it has not found any matching header with content in
  base array
    add empty header to the output
concatenate output arrays
```

Figure 3.14: Pass 2: Recreate matching array.



### 3.2.7 Implementation

DokuWiki is written in PHP. To begin the implementation process it is worth to see documentation page about coding style<sup>7</sup>. There are several obvious reasons to have unified style of code. I chose "vim" as an editor.

To create DokuWiki plugin it is needed to obtain a proper skeleton. It can be generated by the DokuWiki Plugin Wizard<sup>8</sup>.

Because the architecture of this tool is DokuWiki plugin, it is important to know not only a PHP programming language, but also DokuWiki framework. To get familiar with creating plugins it is nice to review already existing plugins<sup>9</sup>. The DokuWiki Development API Reference<sup>10</sup> is also a good place to look during the implementation. It is created by PHPXref<sup>11</sup> which is a nice developer tool itself.

---

<sup>7</sup>[http://www.dokuwiki.org/devel:coding\\_style](http://www.dokuwiki.org/devel:coding_style)

<sup>8</sup><http://dwpluginwiz.chimeric.de>

<sup>9</sup><http://www.dokuwiki.org/plugins>

<sup>10</sup><http://dev.splitbrain.org/reference/dokuwiki/nav.html?index.html>

<sup>11</sup><http://phpxref.sourceforge.net>

## WEAKTYPE PLUGIN

## Chapter 4

# Validation and Use-Cases

### 4.1 Scenario 1 - Web development

We have an imaginary web development company named WebDevel. The company is using DokuWiki with a prototype of WeakType plugin to collect and manage all the data involved in development process. Normal life-cycle of project related knowledge has a beginning similar to described below:

1. Client makes contact with WebDevel
2. WebDevel makes an appointment and creates "Brief" as single Weaki page.
3. Brief page is compared by WeakType plugin with existing types in system.

In this scenario WeakType plugin has types listed bellow:

- auction
- business card
- client
- e-learning
- ecommerce
- forum
- portal
- portfolio
- project
- task

- use\_case
- web\_page

#### 4.1.1 Use-Case 1

Our first use case concerns bookstore company, which get contact with WebDevel to order solution of selling books in the Internet. Info Books, as is the name of the company, has 10-year tradition and decided to widen its horizons in the world of Internet, but decided to spend only a little money for the beginning. Info Books suggested and asked about solution of making not the whole website, but single pages with books descriptions and possibility of selling them. Most of books to sell are old and unique in single copies.

WebDevel worker at arranged meeting with Info Books, as a new client, gathered information and prepared a brief from the meeting, which can be seen in the figure 4.1. It presents the description of company's various data to present in the Internet targeted to sell products in the proper designed layout. It also contains description of sample product.

Brief consists of 5 sections.

1. Contact data
2. Clients - characterize company and its clients.
3. Content lists menu components.
4. Products section presents all kinds of information about product as unified sample product to put on the page and technical selling details.
5. Graphics is a description of page layout with suggestions about colors, pictures and atmosphere to create. It has attached sample product picture to put on the page.

Such brief is not clear, because client do not know exactly what he wants. He asks for advice, prescribes ideas and expectations. Now, it is up to WebDevel to figure out what kind of project offer to client. Having that brief WebDevel can use WeakType plugin to get suggestions what kind of project matches the most to these requirements. WebDevel has a list of project types it can produce.

Plugin says that the best match is "auction". It seems to be a good idea, because auction services already exist in the Internet and designing the single product pages is sufficient for selling them with reduced costs. Info Books can order an auction page as a part of auction service, for example, allegro.pl. It is the biggest auction service in Poland offering auctions of single products put out by company or physical person. Each product has its page developed by seller with description, photos and all the information for transaction needed. Plugin's additional action is sorting the brief according to order in "auction" type. Data are organized.

How plugin did it? Brief consists of 5 sections describing main areas formed on the basis of client needs: Contact, Clients, Content, Products and Graphics with Photos. All these headers are also listed in the project type called "auction", because auction project requires describing these sections. Of course, there are other project types which match here like "e-commerce". Needs presented by Info Books are sufficient for creating an online shop. This is why this proposition appeared after using this plugin. However, Info Books is not ready for ordering an online shop. Less matching types are: portal and portfolio, which also appeared here because of similar content.

#### **4.1.2 Use-Case 2**

This case concerns another client, Domiforte. This company is an intermediary in selling, buying and renting residential properties. WebDevel has made for them business card website few months ago. Brief from this time improved by "business card" type can be seen in the figure [4.2](#).

Few months later, Domiforte contacted with WebDevel to order an Internet portal which should present offers of luxury properties in prestigious locations. Thanks to the business card website, company has developed during these months. Portal is intended to ensure the highest quality of presentation of offers and contact with customers. Available offers must always be current.

This case is different from use-case 1, mainly because client knows exactly what he wants and cooperation has already taken place - this case is an extension of existing project.

Present business meeting consists in creating technical specification of the project, portal appearance and content description on the basis of previous website. WebDevel can reach a composed brief from [4.2](#) above and choose option "Generate" at the "portal" position. In this way, plugin WeakType will generate the data structure needed to fill in order to produce a portal (figure [4.3](#)). Portal specification's sections has been added.

Now, during the meeting, new data can be clearly added and old data refreshed (figure [4.4](#)).

# Validation and Use-Cases

**[[playground:val1\_auction]]** MICHAŁ KACPRZYK THESIS

Edit this page | Old revisions | Recent changes | Search

Trace: » business » e-learning » ecommerce » forum » portal » portfolio » web\_page » validate » auction » val1\_auction

**WeakType plugin data**

Type	Match	Fusion
auction	6	Generate
ecommerce	5	Generate
portal	3	Generate
portfolio	3	Generate
business	3	Generate
e-learning	2	Generate
forum	2	Generate

**Table of Contents**

- Books page
- Contact
- Clients
- Content
- Menu
- Products
- Products categories:
- Selling types:
- Sample book:
- Others:
- Graphics
- Photos

## Books page

### Contact

- Info Books
- info@books.com.pl
- tel. +48 888 888 888

### Clients

Info Books is a books selling company with 10-year tradition. Now, they widen their horizons gaining the Internet market. They are pleased that people are interested to be their clients through this new for them kind of media.

### Content

### Menu

- About company
- View all products
- Add product to favorite
- Ask seller
- Terms and Conditions
- Payment
- Delivery
- Contact

### Products

#### Products categories:

- Esoterics
- Astrology
- Numerologic

#### Selling types:

- Sale at a given price
- Auction

#### Sample book:

Jarosław Gronert "Astrology from the Beginning" book.

#### Features:

- 1 edition, 21.03.07, Pabianice
- Publisher: Jarosław Gronert
- ISBN: 978 83 924948 0 5
- Info: 255 pages + cd
- Price: 12 Euro

#### Author description:

Jarosław Gronert is an astrologer for above 22 years. He has already published two books: "Astrology from the beginning" and "Cosmic Strategy".

#### Book description:

"In nature there have always been present rhythms and cycles constantly influencing on our planetary existence. Unfortunately almost all human interferences in the nature cause disastrous, for it and themselves, consequences. Asking a question if a homo sapiens is able to protect the planet against them may help us to realize our inextricable connection between a man and their earthly and cosmic environment. The nature without a man will survive, a man without the nature - never will!

Nothing lasts forever, but only alters in repeated cycles of births and deaths what ancient Greeks so gracefully included in their Eleusinian mysteries. Human life is a cosmic symphony, and astrology is its notation."

#### Others:

Post the information:

- "Get shipping free buying 2 books!"
- "Prices displayed on our auctions are for online purchases only"

### Graphics

Graphics should be esoteric with positive accents to ensure appropriate atmosphere. Pictures can be made on the basis of books covers (one attached to the brief). Do not use colors not connected with the book's cover. We want client buying not only this book, but build a loyalty and encourage him to buy some of the rest books.

### Photos

One photo without gallery

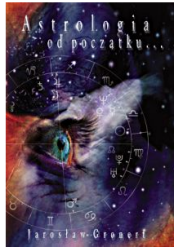


Figure 4.1: Brief from the meeting with client.

**WeakType plugin data**

Type	Match	Fusion
business_card	4	Generate
portal	3	Generate
portfolio	3	Generate
auction	3	Generate
e-learning	2	Generate
forum	2	Generate
ecommerce	2	Generate

**Table of Contents**

- Business Card
- Contact
- Content
- Menu
- Toolbar
- Graphics

---

**Business Card**

**Contact** [Edit](#)

- Domiforte
- info@domiforte.pl
- tel. +48 666 666 666

**Content** [Edit](#)

**Menu** [Edit](#)

- About us
- Properties offers
- Submit offer
- Finance and loans
- Legal advice
- Contact

**Toolbar** [Edit](#)

- Homepage
- Sitemap
- Search

**Graphics** [Edit](#)

Target customers are successful people. Proposed use of logo's color, so purple, because it is slightly associated with royalty, creates a sense of uniqueness. It is cold and mysterious. People who choose the color, often consider themselves to be distinctive from the others. And this is the target consumer.








Figure 4.2: Brief with the structure of "business card" type. Graphics adapted from [Kam09].

**Portal**

---

Specification

---

- Frontpage**
- Document tree**
- Sitemap**
- Administration panel**
- Integration**
- Deadlines**
- Others**

Contact

---

- Domiforte
- info@domiforte.pl
- tel. +48 666 666 666

[Edit](#)

Content

---

**Menu** [Edit](#)

- About us
- Properties offers
- Submit offer
- Finance and loans
- Legal advice
- Contact

**Toolbar** [Edit](#)

- Homepage
- Sitemap
- Search

Graphics [Edit](#)

Target customers are successful people. Proposed use of logo's color, so purple, because it is slightly associated with royalty, creates a sense of uniqueness. It is cold and mysterious. People who choose the color, often consider themselves to be distinctive from the others. And this is the target consumer.






Figure 4.3: Bussines card page converted into portal type structure.



## Portal

---

### Specification

---

#### Frontpage

Three columns. The first two columns of equal width, the third of the width of 200 pixels.

#### Document tree

After entering, you receive a list of articles without a specific category chosen. The sequence can be determined:

- newest
- oldest

#### Sitemap

Standard version with all public documents.

#### Administration panel

- Add / edit / delete articles
- The possibility of accurately determining the priority category
- The possibility of artificial modification of the author and date of publication of the article
- The ability to change the color of links or fonts

#### Integration

Search content through a cloud of tags.

#### Deadlines

- 13.03 - Homepage graphics
- 15.03 - Subpages graphics
- 21.03 - articles / news
- 26.02 - Integration
- 05.04 - Bug fixes, tuning

#### Others

Polish language version

### Contact

---

- Domiforte
- info@domiforte.pl
- tel. +48 666 666 666

### Content

---

Edit

#### Menu

- About us
- Properties offers
- Submit offer
- Finance and loans
- Legal advice
- Contact

Edit

#### Toolbar

- Homepage
- Sitemap
- Search

Edit

### Graphics

---

Edit

Target customers are successful people. Proposed use of logo's color, so purple, because it is slightly associated with royalty, creates a sense of uniqueness. It is cold and mysterious. People who choose the color, often consider themselves to be distinctive from the others. And this is the target consumer.



Figure 4.4: Finished brief of portal page.

## 4.2 Scenario 2 - Migration of the documentation

In this scenario we have an imaginary development company. It uses IBM Mainframe technology along with COBOL programming language, DB2 database and CICS transaction system. The main task of the company is to maintain and improve one existing complex system. All documentation about the system was in 15 000 MS World documents. They were written during activity of the company in particular teams of developers. There was no unified structure of the documentation between the teams. One day, company's manager during browsing a Wikipedia become interested in benefits of wikis. He analyzed documentation system used in the company and decided to create the new, better one. New system is based on DokuWiki with WeakType plugin.

All documents were migrated from MS World format to DokuWiki by automatic tool. It has already been enthusiastically adopted by employees regardless from some unavoidable migration errors. But still, documentation had no unified structure. Searching, navigating and creating additional helpful extensions was difficult. Now, WeakType plugin comes to play.

After analyzing components of the system analyst created types listed bellow:

- CICS transaction
- COBOL batch
- COBOL copybook
- COBOL online
- ControlM definition
- Data set
- DB2 table
- JCL
- REXX script

Now, unifying the structure of pages becomes much easier. It can be done by developers in few short steps:

1. Open not unified documentation page
2. Decide what is the page type
3. Use WeakType plugin to convert page to desired type
4. Improve the result of conversion

Simple example of this procedure is shown in the figures [4.5-4.7](#).

## Validation and Use-Cases

[[playground:val\_jcl]] MICHAL KACPRZYK THESIS

Edit this page Old revisions Recent changes Search

Trace: » cobol\_copybook » data\_set » controlm\_definition » controlm\_definition » cobol\_batch » rexx\_script » cics\_transaction » validate » jd » val\_jcl

**WeakType plugin data**

Type	Match	Fusion
controlm_definition	2	Generate
data_set	2	Generate
rexx_script	2	Generate
cobol_copybook	1	Generate
db2_table	1	Generate
jcl	2	Generate
cics_transaction	1	Generate
cobol_batch	2	Generate
cobol_online	2	Generate

**Name**

Replacement Edit

**Description**

This JCL generates output dataset 555097.OUTPUT.FIX80 from input dataset S55097.INPUT.FIX80. Use program IEBPTPCH to replace columns in sequence 3→1, 2→3;1→2 putting 2 spaces between the records.

Figure 4.5: Example of not unified documentation page.

[[playground:val\_jcl]] MICHAL KACPRZYK THESIS

Edit this page Old revisions Recent changes Search

Trace: » controlm\_definition » controlm\_definition » cobol\_batch » rexx\_script » cics\_transaction » jd » project » val1\_auction » validate » val\_jcl

**JCL**

**Table of Contents**

- JCL
- Name
- Description
- Dependencies
- Programs used
- DB tables used
- Input data sets
- Output data sets
- WeakType not match data

**Name** Edit

Replacement

**Description** Edit

This JCL generates output dataset 555097.OUTPUT.FIX80 from input dataset S55097.INPUT.FIX80. Use program IEBPTPCH to replace columns in sequence 3→1, 2→3;1→2 putting 2 spaces between the records.

**Dependencies** Edit

**Programs used** Edit

**DB tables used** Edit

**Input data sets** Edit

**Output data sets** Edit

**WeakType not match data** Edit

Figure 4.6: Page with the structure of "JCL" type.

[[playground:val\_jcl]] MICHAL KACPRZYK THESIS

Edit this page Old revisions Recent changes Search

Trace: » controlm\_definition » controlm\_definition » cobol\_batch » rex\_script » cics\_transaction » jcl » project » val1\_auction » validate » val\_jcl

### JCL

<b>Name</b> <a href="#">Edit</a>	<b>Table of Contents</b> ▲
Replacement	• JCL
	• Name
	• Description
	• Programs used
	• Input data sets
	• Output data sets
<b>Description</b> <a href="#">Edit</a>	
Replace columns in sequence 3→1, 2→3; 1→2 putting 2 spaces between the records.	
<b>Programs used</b> <a href="#">Edit</a>	
IEBPTPCH	
<b>Input data sets</b> <a href="#">Edit</a>	
S55097.INPUT.FIX80	
<b>Output data sets</b> <a href="#">Edit</a>	
S55097.OUTPUT.FIX80	
	<a href="#">Edit</a>

Figure 4.7: Improved structured documentation page.

## Chapter 5

# Conclusion

### 5.1 Results

The subject "Software Knowledge Management using Wikis" is wide and obviously can be investigated much deeper. In this thesis the main software engineering activities and models were presented with observation that modern models come from combining and improving traditional ones. The knowledge management was presented as modern discipline based on management, philosophy and humanistic sciences. The term of knowledge was not easy to define because of its origin - complex human brain. However, it was possible to present some useful models related to knowledge. The review of literature gives us the main knowledge strategies - codification and personalization. It was shown that the tools and conceptual models of knowledge management can be effectively used in software engineering activities. The reverse sentence is also true, software engineering tools are very useful in knowledge management. Both codification and personalization can be supported by proper software. Presented advantages of wiki systems showed that they can be used in a various applications, particularly as a software knowledge management tools. The wikis were distinguished into traditional and semantic wikis. In the terms of structure restrictions traditional wikis have no restrictions at all, on the other side, semantic wikis puts a lot of pressure on the data structure.

In the practical part of the thesis the prototype of DokuWiki plugin was designed and built. The plugin implements elements of WeakType conception presented in the [CFFA09]. It can be said that it fits in the place between traditional wiki and semantic wiki. As chapter 4 shows, this prototype can be useful in the real applications.

### 5.2 Future work

Like it is almost always in software world, the plugin can be extended much further. In the terms of architecture it can evolve from actual syntax plugin into the action plugin.

## Conclusion

Type	Match	Fusion
business_card	3	Generate
portal	4	Generate
portfolio	3	Generate

- SE
- KM
- KMSE
- wiki

Figure 5.1: Idea of interface improvement.

So it can be active all the time without a need to activate it by inserting [WeakType] mark. This change implies changes in the interface design. Presentation of the matching types can be done as in the "fold unfold list" similar to the wiki page menu. This idea is presented in the figure 5.1.

In the terms of algorithm, both core components can be evolved. Finding the matching types can be improved by using text mining techniques. Creating type based version of the page can be improved by adding support to other kind of type structure elements like empty tables etc.

All in all, we are living in a rapidly evolving world. Thirty years ago there was no WWW, and now, it is used virtually everywhere. Fifteen years ago there was no wiki, now there is the Wikipedia with 19.738.652 pages. It was never so easy to share the knowledge like it is nowadays. Still we have a lot of things to improve. New achievements bring new challenges.

# References

- [AdOdSBD07] N. Anquetil, K.M. de Oliveira, K.D. de Sousa, and M.G. Batista Dias. Software maintenance seen as a knowledge management issue. *Information and Software Technology*, 49(5):515–529, 2007.
- [BD08] F.O. Bjørnson and T. Dingsøy. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50(11):1055–1068, 2008.
- [Boe86] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, 1986.
- [CFFA09] F.F. Correia, H.S. Ferreira, N. Flores, and A. Aguiar. Incremental Knowledge Acquisition in Software Development Using a Weakly-Typed Wiki. 2009.
- [Cun94] Ward Cunningham. *Wiki Design Principles*. <http://c2.com/cgi/wiki?WikiDesignPrinciples>, 1994. Accessed 1 February, 2010.
- [dok10] dokuWiki. *Plugin Development*. <http://www.dokuwiki.org/devel:plugins>, 2010. Accessed 1 February, 2010.
- [DP00] Thomas H. Davenport and Laurence Prusak. *Working knowledge*. Harvard Business Press, 2000.
- [Fre07] Free Software Foundation, Inc., <http://www.gnu.org/licenses/gpl-3.0.txt>. *GNU GENERAL PUBLIC LICENSE Version 3*, 2007. Accessed 1 February, 2010.
- [Her09] Ivan Herman. *Introduction to the Semantic Web*. W3C, <http://www.w3.org/2009/Talks/0615-SanJose-tutorial-IH/Slides.pdf>, 2009. Accessed 1 February, 2010.
- [HNT05] M.T. Hansen, N. Nohria, and T. Tierney. What’s your strategy for managing knowledge? *Knowledge Management: Critical Perspectives on Business and Management*, 77(2):322, 2005.
- [IBM10] IBM, <http://jazz.net/about/about-jazz-vision.jsp>. *Jazz is an IBM Rational initiative to help make software delivery teams more effective.*, 2010. Accessed 1 February, 2010.

## REFERENCES

- [IEE04] IEEE Computer Society. *Software Engineering Body of Knowledge (SWEBOK)*. Angela Burgess, EUA, 2004.
- [Kam09] Paula Kamińska. Wizerunek firm w internecie, 2009. Bachelor thesis, Academy of Humanities and Economics in Lodz.
- [Mic10] Microsoft, <http://msdn.microsoft.com/en-us/teamssystem/dd408382.aspx>. *Team Foundation Server Home*, 2010. Accessed 1 February, 2010.
- [NB07] S. Nerur and V.G. Balijepally. Theoretical reflections on agile development methodologies. *Communications of the ACM*, 50(3):83, 2007.
- [NR69] P. Naur and B. Randell. Software engineering-report on a conference sponsored by the nato science committee. Nato, 1969.
- [NT95] I.A. NONAKA and H.A. TAKEUCHI. *The knowledge-creating company: How Japanese companies create the dynamics of innovation*. Oxford university press, 1995.
- [PRR00] Gilbert Probst, Steffen Raub, and Kai Romhardt. *Managing Knowledge - Building Blocks for Success*. John Wiley & Sons, 2000.
- [RKd03] P.N. Robillard, P. Kruchten, and P. d’Astous. *Software engineering process with the UPEDU*. Addison-Wesley, 2003.
- [RL02] I. Rus and M. Lindvall. Knowledge management in software engineering. *IEEE software*, pages 26–38, 2002.
- [SAA99] G. Schreiber, H. Akkermans, and A. Anjewierden. *Knowledge engineering and management: the CommonKADS methodology*. the MIT Press, 1999.
- [Sch09] K. Schneider. *Experience and Knowledge Management in Software Engineering*. Springer-Verlag New York Inc, 2009.
- [SDHM03] T. Stalhane, T. Dingsoyr, G.K. Hanssen, and N.B. Moe. Post mortem-an assessment of two approaches. *Lecture notes in computer science*, pages 129–141, 2003.
- [Som07] I. Sommerville. *Software Engineering*. Addison-Wesley, Reading, MA, 8th edition, 2007.
- [Sou05] A. Souzis. Building a semantic wiki. *IEEE Intelligent Systems*, pages 87–91, 2005.
- [Sta94] R. Stata. Organizational learning—the key to management innovation. *The training and development sourcebook*, page 31, 1994.
- [Wei85] K. Weick. Cosmos vs. chaos: Sense and nonsense in electronic contexts. *Knowledge in organizations*, pages 213–226, 1985.