Wie wir coden wollen: Zum strategischen Umgang mit der Ressource Wissen in der Softwarearbeit

Tischberger, Roman

Veröffentlichungsversion / Published Version
Zeitschriftenartikel / journal article

WIE WIR CODEN WOLLEN. ZUM STRATEGISCHEN UMGANG MIT DER RESSOURCE WISSEN IN DER SOFTWAREARBEIT

Roman Tischberger

Willkommen im Büro. Die Räumlichkeiten eines süddeutschen Softwaredienstleisters wurden zu Beginn des Jahres 2018 neu bezogen und zeitgemäß eingerichtet. Innenarchitekt*innen haben das Büro dem Geschmack der Softwarebranche und dem, was gegen Ende des Jahrzehnts unter dem Schlagwort *New Work*[1] als modern gilt, angepasst. Das Stockwerk auf einer Industriefläche ist als Großraumbüro konzipiert, elektrisch höhenverstellbare Tische stehen in Gruppen zusammen, nur wenige Bereiche wie Meeting- oder Lagerräume sind durch Türen abgetrennt. Seit November 2017 sind diese Arbeitsplätze Teil einer Arbeitskulturenforschung, die sowohl die soziale Konstruktion von Software als auch die kontinuierliche Aushandlung des Phänomens >Arbeit< in einem mittelständischen Betrieb ethnografisch untersucht und auf der dieser Beitrag beruht. Etwa 120 Beschäftigte, davon etwa 80 Software-Entwickler*innen, bieten auf Projektbasis Softwaredienstleistungen aller Art für Kund*innen überwiegend aus Industrie und freier Wirtschaft an. Die Firma hat, je nach Auftragslage, acht bis zwölf parallellaufende Kundenprojekte; dazu kommen Arbeiten an internen Produkten, die zwischen Entwicklungsbeginn und Marktreife stehen. Die Entwickler*innen sind festen Teams zugeordnet, die gleichzeitig ein oder mehrere Projekte bearbeiten.

Softwarearbeit ist, betrachtet man die unterschiedlichen Arbeitsschritte, die Entwickler*innen durchführen, zu weiten Teilen Wissensarbeit, auf die sich die folgenden Überlegungen beziehen.[2] Programmcode wird denkend entwickelt, auftretende Probleme auf Grundlage von Ideen gelöst. Beobachtbare, praktische Arbeit besteht zu den größten Teilen aus kommunikativen Aushandlungen wie Diskussionen oder Mensch-Maschine-Interaktionen. Schließlich wird Software als Produkt selbst vorwiegend als immateriell-virtuell wahrgenommen. Physisch vermittelt wird sie ausschließlich durch technische Interfaces.

1 *Wolfgang Jäger/Kai-Nils Eicke:* New Work, New Culture, New Leadership. In: Thorsten Petry (Hg.): Digital Leadership. Erfolgreiches Führen in Zeiten der Digital Economy. Freiburg/München/Stuttgart 2019, S. 143–156.

2 *Robert Schmidt:* Praktiken des Programmierens. Zur Morphologie von Wissensarbeit in der Software-Entwicklung. In: Zeitschrift für Soziologie 37 (2008), S. 282–300, hier S. 283–284.

Der Begriff ›Wissensarbeit‹[3] zielt dabei vor allem auf ein Kerncharakteristikum ab: dem Umgang mit und dem Einsatz von Wissen. Sie ist oftmals projektförmig organisiert sowie in Dienstleistungsverhältnisse eingebettet. Der US-amerikanische Literaturwissenschaftler Michael Hardt und der italienische Philosoph Antonio Negri konkretisieren die Grundidee »dieser auf Wissen basierenden Jobs« durch den Begriff der »immateriellen Arbeit«, der jedoch nicht deckungsgleich ist. Deren Konzept beschreibt »eine Arbeit, die immaterielle Güter wie Dienstleistungen, kulturelle Produkte, Wissen oder Kommunikation produziert«.[4] Hardt und Negri gehen bei ihrem Konzept immaterieller Arbeit explizit auf den Computer ein, der als »universelles Werkzeug« für verschiedenste Tätigkeiten eingesetzt werden kann und damit »abstrakte Arbeit« fördert.[5] Bei dieser Betrachtung bleibt jedoch ausgeblendet, dass der Computer allein noch kein universelles Werkzeug ist. Erst das Computerprogramm, die Software, lässt das in der Maschine verankerte Potenzial von Universalität zu. Legt man das Beispiel der Software-Entwicklung an, konstatieren Hardt und Negri zurecht, dass die Arbeit des Programmierens nicht ausschließlich immateriell rund um die Ressource Wissen vollzogen wird, sondern konkrete physische Aspekte, wie das Tippen von Codezeilen vermittels einer Tastatur, untrennbar verwoben bleiben.

Diesen praxisorientierten Aspekt versucht der Soziologe Robert Schmidt in seinen Untersuchungen zu stärken, indem er die vermeintlich rein geistige Wissensarbeit des Programmierens in ihrer konkreten Praxis des Codeschreibens als »verkörpertes Denkhandeln«[6] untersucht. Die Wissensarbeit ›Software-Entwicklung‹ äußert sich im praktisch-leiblichen Vollzug, in dem Arbeitsschritte geplant, Vorgehensweisen diskutiert, Skizzen gezeichnet werden, über Peripheriegeräte mit Computern interagiert wird. Wenngleich also das Produkt der Arbeit überwiegend immateriell ist, vollzieht sich der Prozess des Arbeitens nicht ausschließlich geistig. Immer jedoch ist Wissen die Grundlage der Arbeit. Dabei ist es notwendig, einen prozessorientierten Wissensbegriff anzulegen, wie ihn der britische Organisationssoziologe Frank Blackler vorschlägt: »[…] rather than talking of *knowledge*, with its connotations of abstraction, progress, permanency and mentalism, it is more helpful

3 Der Begriff der Wissensarbeit ist seit seiner Konzeptualisierung durch Peter Drucker in den späten 1950er Jahren mit verschiedenen Definitionen ausgestaltet worden. Nach wie vor steht das Verhältnis von körperlich-physischer Arbeitstätigkeit und der Anwendung von Wissen im Zentrum, allein die scharfe Trennung zwischen beiden Sphären ist verflüssigt.

4 *Michael Hardt/Antonio Negri:* Empire. Die neue Weltordnung. Frankfurt am Main/New York 2002, S. 302.

5 Ebd., S. 304. Der bei Hardt/Negri von Marx entliehene Begriff der ›abstrakten Arbeit‹ wird hier nicht im Sinne des Tauschwerts einer Ware ausgedeutet.

6 *Robert Schmidt:* Soziologie der Praktiken. Konzeptionelle Studien und empirische Analysen. Berlin 2012, S. 169–170.

to talk about the process of *knowing*.«[7] Ebenso wie Arbeit als praktische Tätigkeit prozessualen Charakter hat, gilt dies nach Blackler auch für Wissen. Wird Wissen nicht mehr nur als statisches Konstrukt gesehen, über das man zu einem bestimmten Zeitpunkt verfügen kann – oder auch nicht – lässt sich die Betrachtung auf Praktiken des Wissens legen. Blackler stellt daher nicht das Wissen als Substantiv ins Zentrum seiner Überlegungen, sondern als Verb, um die Anwendung von Wissen – und damit auch, um den Umgang mit Nicht-Wissen – in den Blick zu bekommen.

Hinter den physischen Äußerungen von Softwarearbeit wie Tippen, Klicken, oder Verbalisieren liegen »innere Vollzüge«[8] des geistigen, wissenden Durchdringens der Arbeit. Individuelle Wissensbestände, die abgerufen, erinnert, aktualisiert, synthetisiert oder auch vergessen werden. Wissen über Abmachungen mit den Auftraggeber*innen, gemeinsame Einigungen mit Kolleg*innen, Funktionsweisen des erstellten Gewebes aus Programmcode, Wissen über die Verwendung von Programmiersprachen, Umgang mit auftretenden Problemen und damit verbundenem Erfahrungswissen oder Praktizieren der vereinbarten Regelungen der Projektarbeit. Bei diesen beispielhaften Tätigkeiten müssen die arbeitenden Individuen auf bestehendes Wissen zurückgreifen, es nutzen. Da Software zu großen Teilen kollaborativ im Team hergestellt wird, kommt Wissensmanagement in diesem Prozess eine zentrale Rolle zu. Im Folgenden wird die von Gertraud Koch und Bernd Jürgen Warneken aufgeworfene Frage »wie Wissen – in dem breiten kulturanthropologischen Verständnis des Begriffs – in Arbeitsprozessen als eine aktiv gemanagte und implizit vorhandene Ressource eingesetzt und verwertet wird […]«[9], am Beispiel der Arbeitspraxis in der Software-Entwicklung nachgegangen. Wie wird dort Wissen angeeignet, aufgebaut, verteilt und stabilisiert – und was geschieht, wenn Wissen fehlt?

Software-Entwicklung bei Team Beta

Das Team Beta, das neben fünf Software-Entwickler*innen aus einer Designerin, einem Projektmanager und einem ›Agile Coach‹[10] besteht, arbeitet gemeinsam

7 *Frank Blackler*: Knowledge, Knowledge Work and Organizations: An Overview and Interpretation. In: Organization Studies 16 (1995), S. 1021–1046, hier S. 1035.

8 *Schmidt*, wie Anm. 6, S. 169.

9 *Gertraud Koch/Bernd Jürgen Warneken:* Wissensarbeit und Arbeitswissen: Zur Ethnografie des kognitiven Kapitalismus. Eine Einleitung. In: dies. (Hg.): Wissensarbeit und Arbeitswissen. Zur Ethnografie des kognitiven Kapitalismus. Frankfurt am Main 2012, S. 11–26, hier S. 13.

10 Dies entspricht einer üblichen Teamzusammensetzung im beforschten Softwareunternehmen. Dabei sind Agile Coaches eine kleine Gruppe von erfahrenen Beschäftigten, die alle Teams begleiten und kontinuierlich dafür sorgen, dass die Prinzipien agiler Softwareentwicklung umge-

Abb.1: Einblick in die Arbeitsplätze von Team Beta. Am Stehtisch werden Wissensbestände in kurzen Besprechungen kollektiviert und aktualisiert. Foto: Tischberger/Softwarefirma, Augsburg.

an der Software eines 3D-Scanners. Pauline[11], 29 Jahre alt, ist eine Entwicklerin im Team, die ich über längere Zeit bei ihrer Arbeit begleitet habe und die bei der folgenden ethnografischen Sequenz im Zentrum steht. Sie sitzt an ihrem Schreibtisch und bearbeitet eine ihr zugewiesene Aufgabe: Eine bestehende Funktion der Berechnung des gescannten 3D-Modells soll erweitert werden. Sie klickt sich durch den Programmcode, um eine bestimmte Stelle zu suchen, von der sie zunächst einige Codezeilen kopiert, an anderer Stelle einfügt, ihren Inhalt anpasst und für ihre Zwecke modifiziert. Schnell klickt sie sich durch das Codegewebe, strahlt Ruhe aus, sie scheint zu wissen, was sie macht. Nach gut zehn Minuten hat sie die Funktion entsprechend erweitert und möchte den geschriebenen Code ausführen, um zu überprüfen, ob er wie erwartet funktioniert. Sie klickt, das Programm läuft, erkennbar am sich rasant generierenden Logfile, und zeigt einen Fehler an. »Was, Problem!«, entgegnet Pauline dem Bildschirm trotzig. Der kopierte und angepasste Code scheint

setzt werden. Sie nehmen die Rolle des Scrum Masters ein, moderieren Meetings und führen die in Scrum vorgesehenen Retrospektiven durch.

11 Bei allen im Folgenden zitierten Forschungsmaterialien sind Eigennamen pseudonymisiert. Die Materialien liegen beim Autor.

entgegen ihrer Erwartung nicht funktionsfähig zu sein, der prüfenden Ausführung durch den Computer hält er nicht stand.[12]

Software ist sowohl hinsichtlich Herstellung als auch Wartung relativ instabil und oftmals defizitär. Sie tendiert als komplexes Gewebe aus Maschinenbefehlen dazu, fehlerbehaftet zu sein und sich vor allem während der Entwicklung oftmals entgegen den menschlichen Erwartungen zu verhalten. Je umfangreicher – und damit in der Regel undurchsichtiger in ihrer Struktur – Software wächst, desto wahrscheinlicher ist es, dass sich Fehler bilden oder Instanzen des Programms sich gegenseitig blockieren. So sehen sich Software-Entwickler*innen während ihrer Arbeit regelmäßig damit konfrontiert, dass ihre Änderungen am Code in der Ausführung nicht die erwarteten Ergebnisse bringen. Fehler treten auf, Situationen werden problematisiert und sollen behoben werden.[13] Zudem zeigen sich Probleme nicht ausschließlich softwareseitig, wie es bei Pauline der Fall ist. Der gesamte Entwicklungsprozess von Software unterliegt zahlreichen möglichen Fehlerquellen: Von den Aushandlungen mit den Kund*innen über den erwünschten Funktionsumfang, die Klärung im Team, wie diese in zweiter und dritter Ordnung konstruierten Vorstellungen umzusetzen seien, bis hin zur Programmierung, dem Testen, Ausliefern oder der Wartung des Codes. In allen Situationen der Arbeit können unvorhergesehene Herausforderungen auftreten, die von den Entwickler*innen als Problem identifiziert werden.[14]

Des Widerspenstigen Zähmung. Wissensmanagement in der Arbeit am Code und um den Code

Als primär verfolgter Ansatz zur Linderung der damit verbundenen Probleme zeigt sich die Organisation und Strukturierung der Arbeit, die umso mehr Kooperation, Kommunikation einfordert.[15] Ein für die Projektorganisation übliches Werkzeug sind agile Methoden. Die Bezeichnung ›agile Methode‹, die zunächst im Silicon Valley der späten 1990er Jahre entsteht, aber hauptsächlich im Jahr 2001 mit dem »Agilen

12 Beobachtungsprotokoll Team Beta vom 30. 8. 2018, Abs. 27–29.

13 *Schmidt*, wie Anm. 6, S. 157.

14 Zur Phänomenologie des Fehlers siehe: *Martin Gartmeier* u. a. (Hg.): Fehler. Ihre Funktionen im Kontext individueller und gesellschaftlicher Entwicklung. Münster/New York 2015; *Jean-Claude Laprie:* Dependability: Basic Concepts and Terminology. Wien 1992; *Roman Tischberger*: Computer sagt Nein: Fehlerkulturen in der Softwarearbeit. In: Stefan Groth/Sarah May/Johannes Müske (Hg.): Vernetzt, entgrenzt, prekär? Kulturwissenschaftliche Perspektiven auf Arbeit im Wandel. Frankfurt am Main/New York 2020 (= Arbeit und Alltag. Beiträge zur ethnografischen Arbeitskulturenforschung, Bd. 17), S. 109–129.

15 *Christiane Funken:* Modellierung der Welt. Wissenssoziologische Studien zur Software-Entwicklung. Wiesbaden 2001, S. 45–48.

Manifest«[16] niedergeschrieben wird, meint dabei zweierlei: Einerseits eine Geisteshaltung von Kund*innenorientierung und Flexibilität, die der eigenen Arbeit zugrunde liegt, andererseits unterschiedliche Formen von Projektorganisation. Eine weit verbreitete agile Methode, die im Unternehmen verwendet wird und auf die im Folgenden rekurriert wird, ist >Scrum<. Scrum setzt durch Regeln, Rollenzuweisungen und zeitliche Rhythmen einen organisationalen Rahmen, in den die gesamte Software-Entwicklung eingebettet ist. Große Projekte werden in kleine, planbare Zeiteinheiten, >Sprints< genannt, gegliedert, die jeweils einem festen Ablaufschema folgen. Zu Beginn jedes Sprints werden die anstehenden Aufgaben inhaltlich geplant und festgelegt, die in den folgenden Wochen abgearbeitet werden. Am Sprintende werden die Ergebnisse den Projektverantwortlichen präsentiert. Somit können durch regelmäßige Evaluationen des Projekts die Mitarbeiter*innen relativ flexibel auf Änderungen reagieren und Anpassungen an der Gesamtausrichtung vornehmen. Damit die Teammitglieder wissen, was >am Code< gearbeitet werden soll, wird zunächst Arbeit >um den Code< verrichtet: Mit einem Einblick in die konkrete Feldsituation wird der Prozess der Aufgabenplanung und die Rolle von Wissen darin veranschaulicht.

Wir sitzen im Meetingraum >Ada<, der nach der britischen Softwarepionierin Ada Lovelace benannt ist. Team Beta hält ein >Sprint Planning< ab und bespricht die Arbeitsbausteine für die kommenden drei Wochen. Alle acht Teammitglieder sind da, die Entwickler*innen ebenso wie die Designerin Lea und der >Product Owner< Philipp. Gemeinsam sitzen wir an einem großen Tisch, vor uns ein großer Flatscreen, mit dem Philipps Notebook verbunden ist, so dass alle seinen Bildschirminhalt sehen können. Er öffnet die digitale Aufgabensammlung von Team Beta, das sogenannte >Backlog<, in dem alle Arbeitspakete wie in einem Ticketsystem hinterlegt sind. Nun geht es darum, im Kollektiv einzelne Arbeitspakete, die von Kund*innenseite gewünscht sind, im Detail zu besprechen und gemeinsam festzulegen, was zu tun ist – und schließlich abzuschätzen, wie umfangreich die Erledigung der Arbeit sein wird. Philipp moderiert in seiner Rolle als Product Owner als Bindeglied zwischen Kund*innenvorstellungen und Entwickler*innen-Team und ruft den Task URA-3559 auf, das Projektkürzel und die laufende Nummer des Arbeitspakets. So lässt sich erkennen: Das Projekt ist inzwischen auf eine umfangreiche Größe angewachsen. Die Software soll eine Überprüfung vornehmen, ob bereits generierte 3D-Modelle korrekt angezeigt und vermessen werden, wenn sie nach längerer Zeit wieder aus dem Speicher geladen werden. Im Folgenden ein Ausschnitt des Beobachtungsprotokolls:

> »Mirko meint: Das ging mal, aber das haben wir wohl ausgebaut. Er erzählt, wie es >damals< war. Pauline bremst die Erzählung, es sei nicht ganz so ein-

16 *Kent Beck* u. a.: Manifesto for Agile Software Development (2001). URL: https://agilemanifesto.org/ (Stand: 19. 3. 2020).

> fach, weil eine Measurement-ID fehlt. Mirko glaubt, das führt zu keinem Konflikt und Sascha stützt diese Ansicht. Pauline fasst zusammen und denkt, wie die Änderung noch eingebaut werden könnte und Mikro stimmt zu. Aber ganz klar ist es noch nicht. Sie gehen Schritt für Schritt die Wege durch, was es benötigt, um die gewünschte Wirkung zu erzielen. Nils, der kurz aus dem Meeting gerufen wurde, kommt wieder dazu und fragt nochmals nach dem konkreten Anwendungsfall. Und bringt dann einen neuen Vorschlag, der zunächst gut ankommt, und dann mit >das hatten wir auch gerade andiskutiert< eingegliedert wird. Pauline ist sich ziemlich sicher, dass das >nur von der Platte laden< nicht mehr funktioniert. Mirko dazu: >Ja genau, das hat der Alex ausgebaut, das Fallback.<«[17]

Der Ausschnitt aus dem Feldprotokoll zeigt gemeinsamen, durchaus zähen, Wissensaustausch auf dem Weg zu einer Entscheidung. Die Entwickler*innen Mirko, Pauline, Sascha und Nils diskutieren, schließlich wird noch das ehemalige Teammitglied Alex adressiert. Hinter jeder zu planenden Aufgabe stecken vielschichtige Interpretationen, die sich auf dem Weg zur Umsetzung der Idee in Codezeilen wandeln. Ideen der Kund*innen, die sie verwirklicht sehen möchten, gründen zunächst in deren Interpretation der Software. Diese stellt sich für die Entwickler*innen in Team Beta möglicherweise anders dar, da sie den Code und den konkreten Aufbau der Software kennen. Aber auch innerhalb des Teams existiert heterogenes Wissen über den Code. Im kollaborativen Aushandeln werden schließlich die unterschiedlichen Vorstellungen jedes einzelnen Teammitglieds aktiviert, erinnert, verbalisiert oder auch verschwiegen. Individuelles Codewissen wird gegen bestehendes Erfahrungswissen und mögliche Lösungsansätze abgewogen, im Komplex der Aushandlungen argumentieren die Entwickler*innen über eine als kollektiv sinnvoll erachtete Vorgehensweise. Die jeweils persönlichen, inneren Vollzüge des Wissens >äußern< sich hier im Diskurs, Interpretationen der Aufgabenstellungen werden verbalisiert und so – idealerweise – ein situativer Konsens erzeugt. Im Darüber-Sprechen geschieht Verständigung, persönliche Wissensbestände werden kollektiv aktualisiert und so die Grundlage für gemeinsames, reibungsloseres Arbeiten hergestellt.

Meetings wie dieses Sprint Planning, aber auch andere temporäre Vergemeinschaftungen in der Arbeitspraxis von Team Beta sind, neben dem Aspekt, kommende Aufgaben vorzubereiten, vor allem eine Strategie des Wissensaustauschs. Da die Ressource Wissen überwiegend in individuellen Praktiken angewendet wird,[18] ist kollaborative Wissensarbeit wie Programmieren in höchstem Maße von der Zirkulation von Wissen zwischen den beteiligten Akteur*innen abhängig – und damit dem

17 Beobachtungsprotokoll Team Beta vom 2. 9. 2018, Abs. 11–12.

18 *Schmidt*, wie Anm. 6, S. 169–170.

Management von Wissen. Oder wie Claas, ein Entwickler, es in einem Gespräch zu Beginn der Feldforschung formulierte: »gefühlt 50 Prozent der Arbeit hier im Unternehmen [sind] Kommunikation und Absprache«.[19]

Ist ein Sprint Planning abgeschlossen, werden alle besprochenen Aufgabenpakete schriftlich fixiert und im ›Kanban-Board‹, einer tabellenförmigen To-Do-Liste, visualisiert. Die Arbeit ›um den Code‹ ist vorerst beendet, die Arbeit ›am Code‹ beginnt. Wenn Pauline ein neues Feature einbauen oder eine andere programmiertechnische Änderung vornehmen möchte, muss diese zuvor als Arbeitspaket im Kanban-Board eingetragen sein. Dieser Ticketing-Imperativ erfüllt zwei Zwecke: Übersichtliche Strukturierung und nachvollziehbare Dokumentation, welches Teammitglied welche Arbeitsschritte vollzogen hat. Pauline wählt sich zunächst aus der To-Do-Spalte des Kanban-Boards einen Task aus, der bereits besprochen, geplant und in seinem potenziellen Arbeitsumfang geschätzt wurde. Sie greift nicht nur auf ihr Vorwissen aus den Diskussionen mit ihren Teammitgliedern zurück, sondern vor allem auf ihre individuelle Code-Erinnerung. Diese umfasst nicht den gesamten Programmcode, denn wie bereits Robert Schmidt festhält, gehört es »zu den Hauptschwierigkeiten des Programmierens, die wachsende Codebasis und das sich ständig verändernde Geflecht aus interdependenten Codeteilen im Blick zu behalten«.[20] Dies verdeutlicht ein erneuter Einblick in die zu Beginn angedeutete Programmiersequenz von Pauline:

> »Pauline denkt laut für mich mit und sagt: ›Und jetzt muss ich gucken, wie ich eine »Centration Session« mache. Ich versuche jetzt einfach mal eine zu holen. Beziehungsweise einfach eine zu kopieren und anzupassen. [Pause] Puh. Ist das kompliziert.‹ Sie ändert etwas in der IDE, ihrem Software-Editor-Programm. Der Text wird unterkringelt. ›Was, Problem!‹, spricht sie gegen den Bildschirm. Sie klickt die Fehlermeldung an, sie liest, was der IDE fehlt. Klickt zwei Mal, der Text verändert sich, nichts mehr ist unterkringelt, sie wirkt zufrieden, Problemchen gelöst. Aber das große Problem nicht. ›Wo wird denn das alles gemacht?‹ Sie sucht den Punkt im Code, wo sie die 3D-Daten herbekommen kann, die sie für die Funktion der FrameCustomization verwenden kann. Aber sie findet ihn nicht im Code. Und sucht. ›Boah, keine Ahnung.‹«[21]

Pauline agiert im Rahmen ihrer Wissens- und Nichtwissensbestände, um ihre Programmieraufgabe zu lösen. Zu Beginn schlägt sie vor, einen funktionierenden Code-

19 Beobachtungsprotokoll Gespräch mit Claas vom 27. 11. 2017, Abs. 9.

20 *Schmidt*, wie Anm. 6, S. 167.

21 Beobachtungsprotokoll, wie Anm. 12.

teil zu kopieren und anzupassen, um ihn nicht selbst schreiben zu müssen und dabei eventuell Fehler einzubauen. Eine mögliche Zeitersparnis ist ebenfalls ein Grund, auf bestehendes Wissen zurückzugreifen, das bereits als im Code existente Funktion verankert ist. Pauline kennt diese Funktion und versucht sie zu adaptieren. Dieser Ansatz schlägt jedoch fehl – die Maschine meldet sich in Interaktion mit dem Menschen und gibt ein Signal, das Pauline als bestehendes Problem interpretiert. Solche Situationen kennt sie aus ihrer Programmiererfahrung und weiß, dass es nichts Ungewöhnliches ist, wenn der Computer einen Fehler rückmeldet. Sie nutzt bestehende Strategien des Wissensmanagements, um ihr aufgetretenes Problem zu lösen. Sie ruft zunächst das Logfile, ein Protokoll aller maschinellen Aktivitäten, auf und überprüft, welche Informationen sie dort aus der entsprechenden Fehlermeldung lesen kann. Darüber hinaus bietet die als >IDE< bezeichnete Software-Entwicklungsumgebung weitere Hilfestellungen an, um menschliches Nichtwissen zu managen. Pauline muss sich so weniger merken und weiß das gespannte Sicherheitsnetz aus unterschiedlichen Softwarehilfestrukturen unter sich, das ihre Arbeit >am Code< toleranter gegenüber auftretenden Fehlern macht und ihr in der Konfrontation mit ihrem eigenen Nichtwissen Hilfestellung gibt. Mit der Unterstützung des Logfiles schafft Pauline es, das erste Problem zu lösen, doch im weiteren Verlauf der Programmierarbeit wird sie regelmäßig von weiteren auftretenden Fehlern aufgehalten, die die Lösung des gesamten Tasks unterbrechen. In hermeneutischen Bewegungen des Lösens aufgetretener Probleme nähert sie sich so Stück für Stück der Erledigung ihrer Aufgabe an. Dabei sind die Strategien des Wissensaufbaus nicht ausschließlich auf Software beschränkt. Kolleg*innen zu fragen ist ein gleichermaßen übliches Vorgehen wie über Suchmaschinen und einschlägige Hilfsportale eine weltweite Community von Software-Entwickler*innen im Internet um Unterstützung zu ersuchen. Auch das Projektmanagement mittels Scrum bietet weitere Strategien, um Wissen bei der Arbeit >am Code< zu managen. Ein Beispiel dafür sind Code Reviews: Bevor neu geschriebene Codeteile im Programm aktiv werden, müssen mindestens zwei Teammitglieder den neuen Code ansehen, überprüfen und beurteilen. Neben der Qualitätssicherung durch ein Mehr-Augen-Prinzip wird auch hier – ganz nebenbei – Wissen über den Code zwischen den Teammitgliedern verteilt.

Kollaborative Werkzeuge als Schlüssel des Wissensmanagements

Wie die Beispiele von Team Beta veranschaulichen, hängen viele Problemsituationen der Software-Entwicklung mit Wissen zusammen, mehr noch: sie sind oft Effekt von Nichtwissen. Das Management von Wissen ist daher ein zentraler Weg, um den Logiken von Effizienz und Güte von Arbeit zu entsprechen und Problemen so gut es geht zu begegnen. In der Software-Entwicklung werden unterschiedliche Strategien

genutzt, um Wissen sowohl in der Arbeit ›um den Code‹ als auch in der Arbeit ›am Code‹ zu aktualisieren, stabilisieren, verteilen und zu fixieren.

Das Wissensmanagement greift dabei vor allem auf geteilte Infrastrukturen zurück. Als elementare Rahmenbedingungen von Erwerbsarbeit fixieren, ordnen und verteilen diese das prozessgebundene Wissen der Akteur*innen. Stefan Becks Metaanalyse von Studien zur Kollaboration mittels Papiertechnologien und dem »Schreiben an der Wand«[22] lässt sich problemlos auf virtuelle Werkzeuge der Softwarearbeit übertragen. Denn nicht nur »materielle Umgebungen dienen als gemeinsamer Referenzpunkt und geteiltes Erkenntnisobjekt für Wissensarbeiter, an dem gemeinsame Denkstile und geteiltes stillschweigendes Wissen herausgebildet werden«.[23] Gerade im virtuellen Raum sind die Softwarearbeiter*innen in ein Netz aus mannigfaltigen Referenzpunkten und softwareseitigen Lösungen für kollaboratives Arbeiten eingebunden. Beispielsweise bei Tätigkeiten ›um den Code‹, wie der Nutzung von Backlog und Kanban-Board, die die ausgehandelten Arbeitspakete fixieren oder Wiki-Systemen, in denen besprochene Meetinginhalte dokumentiert werden. Auch das konkrete Arbeiten ›am Code‹ wird durch Tools unterstützt: Versionsverwaltungen, die verteiltes Programmieren an gemeinsamer Codebasis erlauben, Plattformen, die Qualitätskontrollen des Codes koordinieren oder schließlich die zahlreichen eingebauten Hilfsmittel der IDE, von Autovervollständigung über Debugger bis zu Logfiles. Doch auch nichtvirtuelle Materialien werden in der diskursiven Wissenspraxis genutzt, oftmals in digital-analoger Verschränkung. Zwar werden Stift und Papier bei Team Beta nur selten verwendet, um Programmroutinen manuell-zeichnend zu durchdenken,[24] gemeinsame Diskussionen am Whiteboard sind jedoch fester Bestandteil der Arbeitsroutine.

Beide Sphären der Software-Entwicklung, das Aushandeln wie das Programmieren, bedingen sich gegenseitig und überlappen sich in der Praxis: Die situativen Aushandlungen der Arbeitspakete im Planning werden digital im Backlog festgehalten, ein Programmierfehler kann in einem der regelmäßigen Meetings mit Kolleg*innen besprochen werden. Gemeinsame Diskussionen am Whiteboard, deren Ergebnisse fotografiert und digital hinterlegt werden, sind ebenso wie der große, mit Konferenztechnik ausgestattete Stehtisch, an dem kurze Tagesbesprechungen abgehalten und parallel am Bildschirm das Kanban-Board gepflegt wird, Beispiele für analog-digital verschränkte Wissenspraxis.

22 *Stefan Beck:* Anmerkungen zu materiell-diskursiven Umwelten der Wissensarbeit. In: Gertraud Koch/Bernd Jürgen Warneken (Hg.): Wissensarbeit und Arbeitswissen. Zur Ethnografie des kognitiven Kapitalismus. Frankfurt am Main 2012, S. 27–39, hier S. 34–35.

23 Ebd., S. 36.

24 *Schmidt*, wie Anm. 6, S. 167–173.

Alle genutzten Strukturen haben neben ihrem Hauptzweck als Werkzeug der Arbeitsorganisation einen entscheidenden Nebeneffekt: Wissen über den Code, das immer subjektiv und situativ erzeugt wird, im gemeinsamen Arbeiten ständig zu verteilen und aktuell zu halten. Dennoch: Bei allen Infrastrukturen >am< und >um den Code< ist letztlich die Praxis des Nutzens, Umnutzens oder Nichtnutzens entscheidend. Bei kollaborativer Wissensarbeit wie der Software-Entwicklung ist die Herstellung und Aufrechterhaltung eines gemeinsam geteilten Vorgehens erforderlich. Und selbst dann ist die Arbeit kein rein positiver Prozess: Idealvorstellungen, mit denen die Werkzeuge des Projektmanagements in der Regel verknüpft sind, werden nicht so reibungslos umgesetzt, wie sie in der Theorie funktionieren. Widerständigkeiten und Widersprüchlichkeiten treten in einer Umgebung, in der logische Vorgehensweisen und kybernetisch-strukturalistische Prinzipien erwünscht sind, weiterhin auf und unterliegen dem ständigen Versuch, in bestehende Strukturen integriert zu werden. Dies gelingt jedoch nie vollständig. Als Effekt ist in der Softwarebranche die Leitidee der >Agilität< in Mode gekommen, die diesen Faktor der Unberechenbarkeit des vermeintlich Berechenbaren nicht ausschließt, sondern als Phänomen akzeptiert – und dennoch weiterhin versucht, ihn so weit möglich zu kontrollieren.

Roman Tischberger
Universität Augsburg
Lehrstuhl für Europäische Ethnologie/Volkskunde
Universitätsstraße 10
86159 Augsburg
roman.tischberger@philhist.uni-augsburg.de