Wissensmanagement als integraler Bestandteil des Software Engineerings

Konzeption einer Vorgehensweise unter Einbindung agiler Modelle am Beispiel des Web 2.0-Unternehmens Jimdo

Sönke Ruempler
Matrikelnummer: 541465
Große Brunnenstr. 9
22763 Hamburg
E-Mail: soenke@ruempler.eu

Diplomarbeit, vorgelegt zur Erlangung des Zeugnisses über die Diplomprüfung im Studiengang »Diplom Wirtschaftsinformatik (FH)« der staatlich anerkannten Fachhochschule AKAD, Pinneberg.

Betreuender Fachhochschullehrer: Prof. Dr. Roland Schwesig

Inhaltsverzeichnis

Abkürzungsverzeichnis

ATDD . . . . . Acceptance Test-driven Development

CFD . . . . . Cumulative Flow Diagram

CI . . . . . Continuous Integation

CoP . . . . . Community of Practice

DSL . . . . . Domain specific language

IEEE . . . . . Institute of Electrical and Electronics Engineers

SbE . . . . . Specification by Example

SEC . . . . . Software Experience Center

SOA . . . . . Service oriented architecture

SWEBOK . . . . . Guide to the Software Engineering Body of Knowledge

TDD . . . . . Test-driven Development

TFD . . . . . Test-first Development

WIP . . . . . Work in Progress

XP . . . . . Extreme Programming

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problemstellung, Motivation und Zielsetzung

In der vorliegenden Diplomarbeit wird das Thema »Wissensmanagement als integraler Bestandteil des Software Engineerings« vor dem Hintergrund eines agil[1] orientierten Web 2.0-Unternehmens betrachtet. An dem konkreten Beispiel des Unternehmens Jimdo wird eine Vorgehensweise für die Etablierung eines Wissensmanagements für Teams im Software Engineering konzipiert, die auf den bereits im Unternehmen eingesetzten agilen Softwareentwicklungsmethoden aufsetzt.

Softwareentwicklung ist ein hochkomplexes und wissensintensives Feld.[2] In der heutigen Wissensgesellschaft ist die Ressource Wissen zur dominanten Produktivkraft geworden[3], somit müssen Organisationen und ihre Mitglieder ständig dazulernen. Sie müssen aber auch lernen, zu vergessen und »ihr Verhalten im Lichte neuer Kenntnisse und Einsichten teilweise zu revidieren«[4]. Für eine lernende Organisation sind dies selbstverständliche Werte. Systemisches Wissensmanagement und agile Softwareentwicklung setzen hier an: Sie geben sich nicht nur mit der Unsicherheit und Komplexität der Umwelt ab, sondern erwarten diese und versuchen, die Organisation so flexibel und reaktionsfähig zu halten, dass sie ihre Innovationskraft nachhaltig aufrecht erhalten können.[5]

Die Motivation zu dieser Arbeit entstand einerseits aus dem persönlichen Interesse des Verfassers an der Verbindung von Wissensmanagement und agiler Softwareentwicklung, auf der anderen Seite wurde durch eine Kurzumfrage bei Software Teams von Jimdo festgestellt, dass Bedarf für Wissensmanagement besteht, gerade in Bezug auf Problematiken wie den Austausch zwischen Teams und die Aktuellhaltung von Dokumentationen.

»Traditionelle« bzw. dokumentenorientierte Ansätze des Wissensmanagements haben gewisse Grenzen, wie im Laufe der Arbeit gezeigt wird. Ihr Einsatz eignet sich nicht für ein Unternehmen wie Jimdo, das sich durch flache Hier-

[1] »agil« wörtlich übersetzt bedeutet »von großer Beweglichkeit zeugend; regsam und wendig«, vgl. URL: `http://www.duden.de/rechtschreibung/agil` (abgerufen am 15.11.2012))
[2] vgl. Bjørnson und Dingsøyr 2008, S. 3.
[3] vgl. Willke 2007, S. 22.
[4] Willke 2001, S. 85.
[5] vgl. ebd., S. 90; vgl. Dybå und Dingsøyr 2008, S. 3.

archien, schnelles Wachstum und Arbeit mit agilen selbstorganisierten Teams definiert und unter einem hohem Konkurrenz- und Innovationsdruck steht.

Die bisherige Forschung im Bereich Wissensmanagement und Software Engineering unter Berücksichtigung agiler Methoden ist gering. Größtenteils werden in Bezug auf Software Engineering technokratische Lösungen zur Wissensmultiplikation[6] angeboten, die sich nicht an Unternehmensstrategien ausrichten. Daneben gibt es nur vereinzelte wissenschaftliche Artikel zu agilen Methoden und Wissensmanagement, die das Thema jedoch nur kurz anreißen. Sie sprechen lediglich die Wissensverteilung an und geben keine weitere Systematisierung vor.[7] Levy und Hazzan 2009 sehen einen Forschungsbedarf darin, zu untersuchen, welche und inwiefern agile Vorgehensweisen Wissensmanagement implizit unterstützen, weiterhin plädieren sie dafür, das organisatorische Umfeld und die Unternehmenskultur zu untersuchen.

Daraus leitet sich für die vorliegende Arbeit die Frage ab, inwiefern agile Softwareentwicklungsmethoden schon implizit Wissensmanagement »betreiben«, ohne als explizites Wissensmanagementinstrument installiert worden zu sein? Oder anders: Entscheidet sich ein Unternehmen für den Einsatz agiler Softwareentwicklung, bekommt es dann ein Wissensmanagement »mitgeliefert«?

Dies wird im Verlauf der Diplomarbeit anhand des Unternehmensbeispiels von Jimdo untersucht. Es wird ein Modell für ein Wissensmanagement entwickelt, welches bei den bereits bestehenden agilen Methoden ansetzt und die Vorgehensweise konzeptionell erweitert. Dazu wird folgende Leithypothese formuliert:

Der Einsatz von agilen Softwareentwicklungsmethoden kann implizit für ein ausreichendes Wissensmanagement im Software Engineering sorgen, ohne explizit als strategisches Instrument des Wissensmanagements angewendet zu werden.

Daraus ergeben sich folgende Leitfragen:

- Warum ist Wissensmanagement im Software Engineering wichtig?
- Wie sehen bisherige Ansätze in der Literatur von Wissensmanagement im Software Engineering in der Literatur aus?

[6] vgl. Bjørnson und Dingsøyr 2008, S. 12.
[7] vgl. Chau u. a. 2003; vgl. Melnik und Maurer 2004.

- Sind bisherige Ansätze bei Jimdo einsetzbar?
- Wo unterstützen agile Methoden Wissensmanagement? Wie kann dies untersucht werden?
- Wo sind Grenzen agiler Methoden im Wissensmanagement? Wo müssen traditionelle Methoden des Wissensmanagement ergänzend ansetzen?
- Wie könnte eine Vorgehensweise für ein Wissensmanagement als integraler Bestandteil für Software Teams bei Jimdo skizziert aussehen?

1.2 Rahmen und Vorgehensweise der Untersuchung

In dieser Diplomarbeit wird das Web-2.0-Unternehmen Jimdo betrachtet. Im Unternehmen wird kein explizites Wissensmanagement betrieben, jedoch werden agile Methoden im Bereich des Software Engineerings bereits eingesetzt. Die agile Vorgehensweise ist in der Unternehmenskultur fest verankert. Die Auswahl der agilen Methoden geschieht in Anlehnung an die bei Jimdo bereits eingesetzten Methoden. Da es sich um ein Unternehmen handelt, welches Online-Software entwickelt, geht es in Bezug auf technische Beispiele um Webanwendungen und nicht um sonstige Software, wie z. B. Software, die heruntergeladen oder von einer CD geladen und installiert werden muss.

Die Vorgehensweise in dieser Arbeit verbindet eine praktische Unternehmensrealität mit theoretischen Ansätzen. Vor dem Hintergrund von theoretischen Modellen wird die Unternehmenspraxis eingeordnet und untersucht. Im Hauptkapitel wird auf Grundlage der bisherigen Ausführungen eigenständig ein »Konzept eines Wissensmanagementmodells« entwickelt, welches zum einen auf der Unternehmenspraxis basiert und zum anderen Ergänzungen enthält, die aus der Theorie abgeleitet werden. Das Modell wird somit erweitert und bietet einen Ansatz, um bei Jimdo ein optimales Wissensmanagement zu etablieren. So wird die Fragestellung in dieser Arbeit, inwiefern Wissensmanagement, über den Einsatz von agilen Methoden, bereits integraler Bestandteil des Software Engineerings ist, vor dem Hintergrund des Modells »Wissensmanagement als Geschäftsprozess« von Willke[8] analysiert. Dabei wird zusätzlich zwischen verschiedenen Wissensarten unterschieden, um die klassische Einteilung in implizites und explizites Wissen zu erweitern. Theoretisch zugrunde gelegt wird die Perspektive eines systemischen Wissensmanage-

[8] vgl. Willke 2001.

ments, wie Willke es verfolgt. Die Vorgehensweise bei Jimdo wird nach Schneider[9] einem personenzentrierten Wissensmanagementansatz zugeordnet. Im Hauptteil dieser Arbeit wird eigenständig ein Konzept entwickelt, dass sich an die Ebenen des St. Galler Managementmodells anlehnt und dazu dient, die bei Jimdo eingesetzten agilen Methoden in Bezug auf die Managementebenen (normativ, strategisch, operativ) einzuordnen. Mithilfe der Konzeptentwicklung können mögliche Unzulänglichkeiten agiler Methoden für den (Wissens-)Managementprozess aufgedeckt und ergänzt werden.

Wissensmanagement berührt verschiedene Disziplinen und Ansätze, die im Rahmen einer Diplomarbeit nicht in ihrer Gesamtheit aufgegriffen werden können. Zur sinnvollen Eingrenzung des Themas bleiben folgende Bereiche außen vor: Wissensmanagementtools nach North[10], Datenmanagement, Informationsmanagement, Lerntheorien, Data Warehousing und -mining sowie die Auftrennung der Begrifflichkeiten Daten, Information und Wissen[11].

Ebenso können agile Methoden in dieser Arbeit nicht im Detail und in Bezug auf ihre Vor- und Nachteile behandelt werden. Sie werden vor dem Hintergrund betrachtet, in dem sie in dem Unternehmen Jimdo im Bereich Software Engineering bereits eingesetzt und als Grundlage eines Wissensmanagementmodells genutzt werden können.

Der Begriff »Agile Softwareentwicklung« schließt in dieser Arbeit Lean Software Development Modelle mit ein[12].

1.3 Aufbau der Arbeit

Die Diplomarbeit ist in fünf Abschnitte gegliedert:

- Einleitung
- Wissensmanagement und Software Engineering
- Agile Softwareentwicklung bei Jimdo
- Konzept: Entwicklung eines Wissensmanagementmodells sowie
- Fazit und Ausblick.

[9] vgl. Schneider 2001.
[10] Portal-Software, E-Learning Systeme, Groupware, Dokumentenmanagementsysteme, Contentmanagementsysteme, vgl. North 2011, S. 316
[11] vgl. ebd., S. 16 ff.
[12] vgl. Dybå und Dingsøyr 2008, S. 3 f.; vgl. Fowler 2008.

Zur Übersichtlichkeit schließen die Kapitel jeweils mit einer Zusammenfassung und kritischen Betrachtung ab.

Im Kapitel »Wissensmanagement und Software Engineering« werden grundlegende Definitionen getroffen und in das Thema eingeführt:

- Die Begriffe »Wissen und Wissenstypenn« werden differenziert, danach werden »Wissensmanagement und Modelle«, die für das Vorgehen in dieser Arbeit relevant sind, vorgestellt.
- Im Abschnitt »Software Engineering und seine Wissensbereiche« werden Software Engineering und die zugehörigen Wissensgebiete vorgestellt.
- Es werden die »Wissensbereiche« des Software Engineerings benannt und
- »Arten von Wissen«, auf die im späteren eingegangen werden soll, beschrieben und eingegrenzt.
- Es folgt ein Blick auf die »Unterschiede traditioneller und agiler Softwareentwicklung«.
- Zur Orientierung wird ein »Überblick bisheriger Forschungsansätze« zu Wissensmanagement im Software Engineering gegeben sowie eine
- »Kurzvorstellung agiler Softwareentwicklungsmodelle«, die bei Jimdo im Einsatz sind: Scrum, Extreme Programming, Software Kanban, DevOps sowie ergänzende Methoden.

Im Kapitel »Agile Softwareentwicklung bei Jimdo« wird das Unternehmen Jimdo in den Fokus gerückt:

- Zunächst wird Jimdo im Abschnitt »Unternehmensvorstellung« näher betrachtet. Dazu wird »Generelles zum Unternehmen« und ein Einblick in die »Unternehmenskultur« gegeben.
- Der Abschnitt »Software Teams« erläutert zuerst »Struktur und Organigramm« der bestehenden Software Teams. Danach werden in »Eingesetzte agile Methoden« die Methoden vorgestellt, die bei Jimdo bereits Anwendung finden und in dieser Arbeit als Ausgangspunkt für die Modellkonzeption gewählt werden.
- Zum Abschluss dieses Kapitels wird durch eine »Zusammenfassung und normative Einordnung« des Unternehmens das Grundmodell entworfen, welches im Hauptkapitel zum Konzept eines Wissensmanagementmodells vervollständigt wird.

Das Kapitel »Konzept: Entwicklung eines Wissensmanagementmodells« stellt den Hauptteil der Arbeit dar:

- Zur Überprüfung der Leitthese, dass agile Methoden implizit ein ausreichendes Wissensmanagement mit sich bringen, werden im Abschnitt »Einordnung agiler Modelle« die agilen Methoden dahingegen untersucht, inwiefern sie zu Wissensmanagementaktivitäten beitragen können. Die gewählte Systematik lehnt sich an das Modell »Wissensmanagement als Geschäftsprozess« nach Willke an. Zudem werden die, im vorherigen Kapitel bestimmten, Wissensarten in die Einordnung einbezogen.
- Im Abschnitt »Beispiele« werden die Ausführungen in einer Abbildung zusammengefasst, die die Wissensflüsse in agilen Teams skizzieren. Abgeschlossen wird dieser Punkt durch eine »Zusammenfassung und kritische Betrachtung«.
- Im Abschnitt »Überprüfung und Vervollständigung des Modells« werden die Ergebnisse aus den vorherigen Kapiteln zusammengefasst und auf Jimdo angewendet. Um ein für Jimdo optimales Wissensmanagement zu etablieren, wird dabei auf strategischer und operativer Ebene »Wissensmanagement als Geschäftsprozess« ergänzt und angepasst.

Das Kapitel »Fazit und Ausblick« schließt die vorliegende Diplomarbeit ab.

2 Wissensmanagement und Software Engineering

Im diesem einführenden Kapitel werden grundlegende Definitionen und Differenzierungen zum Thema »Wissensmanagement und Software Engineering« getroffen, ein Überblick über bisherige Forschungsansätze gegeben sowie agile Softwareentwicklungsmodelle vorgestellt.

2.1 Wissen und Wissensmanagement

2.1.1 Wissen und Wissenstypen

In der Literatur gibt es zahlreiche Definitionen von Wissen, die aus verschiedenen Richtungen stammen, wie Pädagogik, Soziologie, Philosophie, Neurowissenschaft uvm. Eine allgemeingültige Definition gibt es nicht, denn sie muss vor dem Hintergrund des jeweiligen Kontextes gesehen werden, so Müller.[13] In betriebswirtschaftlichen Zusammenhängen ist eine Begriffsabgrenzung auf der Grundlage der Unterscheidung zwischen Daten, Informationen und Wissen verbreitet.[14] Auch Willke nimmt diese Unterscheidung vor und fasst zusammen: »Daten sind der Rohstoff für alles Wissen. Informationen sind systemspezifisch aufbereitete Daten und damit Zwischenprodukte des Wissens. Wissen ist die Veredelung von Information durch Praxis. Jedes Wissen setzt Praxis voraus.«[15]

Eine weitere Definition von »Wissen« geben Probst u. a., die die Konstitution und Funktion von Wissen beschreiben sowie die individuelle Konstruktion im Entstehungszusammenhang und die Personengebundenheit des Wissens betonen:

»Wissen bezeichnet die Gesamtheit der Kenntnisse und Fähigkeiten, die Individuen zur Lösung von Problemen einsetzen. Dies umfasst sowohl theoretische Erkenntnisse als auch praktische Alltagsregeln und Handlungsanweisungen. Wissen stützt sich auf Daten und Informationen, ist im Gegensatz zu diesen je-

[13] vgl. Müller 2009, S. 25.
[14] vgl. ebd., S. 26.
[15] Willke 2007, S. 28.

doch immer an Personen gebunden. Es wird von Individuen konstruiert und repräsentiert deren Erwartungen über Ursache-Wirkungs-Zusammenhänge.«[16]

Willke ergänzt, dass Wissen im Unternehmenskontext immer zweckgebunden ist und die spezifische Bedeutung mit den Zielen und der strategischen Ausrichtung des Unternehmens verbunden ist.[17] Bei Wissen handelt es sich für ihn um eine »aus Erfahrung gegründete, kommunizierte, konstituierte und konfirmierte Praxis«[18]

Eine weitere Begriffsabgrenzung ist die zwischen Wissen und Nicht-Wissen, also Wissen in Abgrenzung zu seinem Gegensatz.[19] Willke betont in diesem Zusammenhang, dass Wissensmanagement auch immer als Management von Nicht-Wissen beinhaltet. Dazu gehört der Umgang bzw. das in Kauf Nehmen von Ungewissheiten und Unsicherheiten.[20]

Eine grundlegende und in der Literatur häufig aufgegriffene Differenzierung von Wissenstypen nimmt Polanyi vor. Er unterscheidet zwischen implizitem (»tacit«) und explizitem (»explicit«) Wissen. Implizites Wissen hat eine Person aufgrund »ihrer Erfahrung, ihrer Geschichte, ihrer Praxis und ihres Lernens im Sinne von 'know how'«[21], dabei muss es der Person nicht bewusst sein, dass sie es weiß oder kann, und sie muss dies nicht ausdrücken oder erklären können. Explizites Wissen ist der Person bewusst und kann verbalisiert werden.[22]

Nonaka und Takeuchi greifen die Unterscheidung von Polanyi auf und beschäftigen sich mit der Gestaltung der Übergänge zwischen den Wissenstypen. Sie beschreiben »Four modes of knowledge conversion«[23], wie die Abbildung 1 zeigt. Bei der »Socialization« geht implizites in implizites Wissen über. Ausschlaggebend sind geteilte Erfahrungen, wie z. B. wenn ein Lehrling durch Zugucken von seinem Meister lernt ohne dass gesprochen werden muss.[24] Im Prozess der »Externalization« geht implizites in explizites Wissen über. Dabei wird implizites Wissen (sprachlich und/ oder schriftlich) artikuliert und in

[16] Probst u. a. 2010, S. 23.
[17] vgl. Willke 2007, S. 34.
[18] vgl. ebd., S. 33.
[19] vgl. Müller 2009, S. 44; Schneider 2006 differenziert noch zwischen Formen des Nichtwissens, was hier leider aus Platzgründen nicht aufgegriffen werden kann.
[20] vgl. Willke 2007, S. 27.
[21] Polanyi zit. n. Willke 2001, S. 12
[22] vgl. ebd., S. 13.
[23] Nonaka und Takeuchi 1995, S. 62.
[24] vgl. ebd., S. 63.

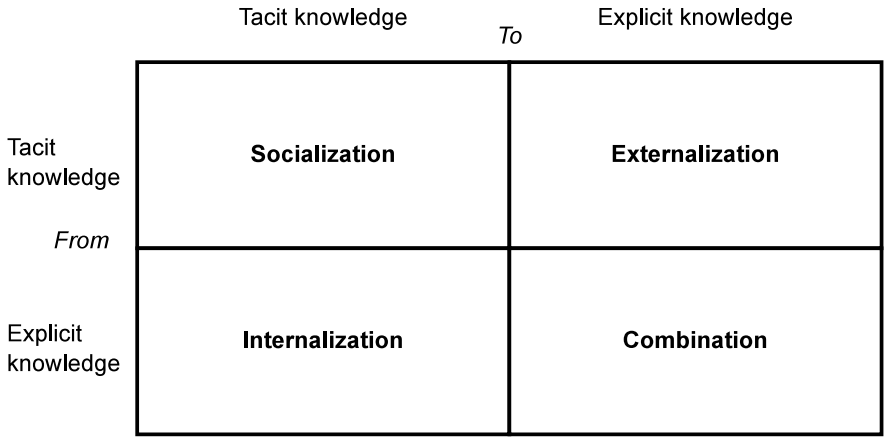

Abbildung 1: Four modes of knowledge conversion (entnommen aus Nonaka und Takeuchi 1995, S. 62)

explizite Konzepte überführt, z. B. in Metaphern, Analogien, Konzepte, Hypothesen oder Modelle. Nonaka und Takeuchi weisen darauf hin, dass dieses Wissen auf individueller Interpretation und Eindrücken basiert und nicht die ultimative Wahrheit darstellt, allerdings trägt es zur Reflexion von und Interaktion zwischen Personen bei.[25] »Combination« meint die Überführung von explizitem zu explizitem Wissen über die Systematisierung von Konzepten in andere Wissenssysteme. Medien dazu sind bspw. Dokumente, Meetings, Telefonate, Computer. Diese Umgestaltung von existierenden Informationen kann zu neuem Wissen führen.[26] Bei der »Internalization« wird explizites zu implizitem Wissen, indem es verinnerlicht und in Routine überführt wird. In diesem Prozess liegt internalisiertes Wissen in Form von geteilten mentalen Modellen oder technischem Know-how vor, was für das Unternehmen eine wertvolle Entwicklung darstellt.[27] In Unternehmen kann innovatives Wissen generiert werden, wenn die Wissensübergänge »in routinierte organisationale Prozesse« gefasst sind, die das Artikulieren, zugänglich Machen und Verbreiten von individuellem Wissen unterstützen.[28]

Willke lehnt sich an den Begriff des »organizational learning« bei Argyris und Schön[29] an und betont den Unterschied zwischen organisationalem und personalem Wissen. »Organisationales oder institutionelles Wissen steckt in den

[25]vgl. Nonaka und Takeuchi 1995, S. 64.
[26]vgl. ebd., S. 67.
[27]vgl. ebd., S. 69.
[28]vgl. Willke 2001, S. 15.
[29]Argyris und Schön zit. n. ebd., S. 16.

personen-unabhängigen, anonymisierten Regelsystemen, welche die Operationsweise eines Sozialsystems definieren.«[30] Dabei handelt es sich z. B. um »Standardverfahren [...], Leitlinien, Kodifizierungen, Arbeitsprozessbeschreibungen, [...] Routinen, Traditionen, spezialisierte Datenbanken, kodiertes Produktions und Projektwissen und die Merkmale der spezifischen Kultur einer Organisation«[31].

2.1.2 Wissensmanagement und Modelle

Vor dem Hintergrund von Willkes Ausführungen zu personalem und organisationalem Wissen, muss sich Wissensmanagement also damit beschäftigen, »wie das Zusammenspiel von personalem und organisationalem Wissen verstanden und organisiert werden kann.«[32] Mit dem Blick auf das Ziel von Organisation, deren wichtigste Produktivressource Wissen ist[33], formuliert Willke:

»Kern von Wissensmanagement [ist] die Fähigkeit [...], die zukünftige Innovationskompetenz der Organisation dadurch zusichern, dass kollektives Lernen und eine kontinuierliche Revision des vorhandenen Wissens aktiviert werden«[34]

Einige grundsätzliche Überlegungen zur systemischen Perspektive seien angemerkt, ohne an dieser Stelle aus Platzgründen die systemischen Hintergründe im Detail zu erläutern: Organisationen sind soziale Systeme und »reproduzieren sich mittels Kommunikation ständig selbst, sind in permanenter Veränderung begriffen und schaffen immer neue Ordnungsgefüge in Form von erinnerter Geschichte, strukturell festgehaltenen Erfolgen und abgestimmten Wahrnehmungsmustern und Erwartungshaltungen.«[35]

Zusammenfassend sagen Königswieser und Hillebrand: »Das Innenleben von Organisationen wird über Reduktion von Komplexität gesteuert, bzw. die Organisation steuert sich selbst über geteilte Sinnbilder, Wertehierarchien und Vi-

[30] Willke 2001, S. 16.
[31] Ebd., S. 16.
[32] Ebd., S. 29.
[33] »Organisationale Wertschöpfung durch Expertise« (ebd., S. 99)
[34] Ebd., S. 90.
[35] Königswieser und Hillebrand 2005, S. 35.

sionen, über Sitten, Rituale und Gebräuche, über Rollenzuteilungen und Hierarchien und vor allem über Objektivierung von Vereinbarungen.«[36]

»Organisationales oder institutionelles Wissen steckt in den personenunabhängigen, anonymisierten Strukturen, Prozessen und Regelsystemen, welche die Operationsweise eines Sozialsystems definieren.«[37] Eine Organisation lernt, indem sie »in ihre Strukturen, Prozesse und Regelsysteme Wissen« einbaut[38]. Darüber sind Organisationen in der Lage, Wissen zu erwerben, welches umfassender und komplexer als das von einzelnen Personen ist[39], und Leistungen erbringen, die über die Leistungen einzelner hinausgehen.

Notwendig ist eine Unternehmenskultur, die »den nötigen Freiraum für Wissensbasierung, Lernfähigkeit und Kontextsteuerung«[40] schafft und damit einhergehende Ungewissheiten aushält.

Eine Steuerung von komplexen sozialen Systemen, wie Unternehmen oder Teams sie darstellen, ist aufgrund der operativen Geschlossenheit und Selbstreferenz von Systemen nicht möglich. Systeme folgen ihrer eigenen Logik und Dynamik und basieren auf ihren eigenen Regeln. Steuerung ist nur über Kontextsteuerung und Selbststeuerung möglich.[41]

Die Frage, wie das Wissen von Personen und Organisationen beidseitig generiert, genutzt und wechselseitig zur Verfügung gestellt werden, so dass das Wissenspotential ausgeschöpft werden kann, beantwortet Willke mit seinem Modell des »Wissensmanagements als Geschäftsprozess«. Wichtige Voraussetzung ist, dass Wissensmanagement als Teil des allgemeinen Managements verstanden wird und als Geschäftsprozess in die Unternehmensstrategie integriert ist.[42]

Abbildung 2 zeigt das Modell des Wissensmanagements als Geschäftsprozess von Willke. Es zeigt den idealtypischen Fluss von Wissen durch eine Organisation. Dabei sind die ersten drei Schritte »Generierung von Wissen«, »Topographien von Wissen« und »Dokumentation von Wissen«. Ein System, also z. B. ein Teammitglied, welches Wissen generiert, kann dieses Wissen nicht selbst-

[36] Königswieser und Hillebrand 2005, S. 32.
[37] Willke 2007, S. 58.
[38] Ebd., S. 59.
[39] vgl. ebd., S. 59.
[40] Ebd., S. 62.
[41] vgl. ebd., S. 25.
[42] vgl. ebd., S. 106.

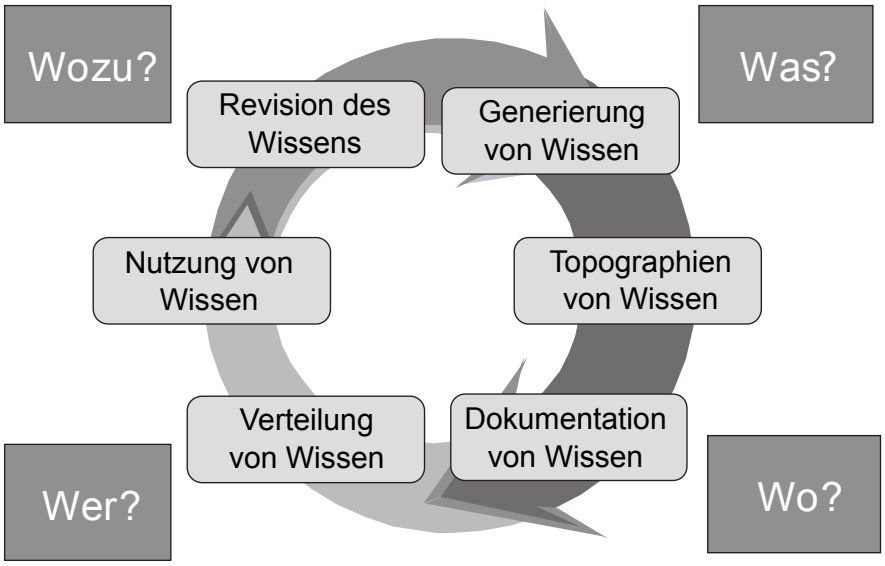

Abbildung 2: Wissensmanagement als Geschäftsprozess (entnommen aus Willke 2001, S. 89)

ständig überprüfen, da es selbstreferentiell ist. Dies könne zur Folge haben, dass die Generierung von Wissen zweckfrei ist. Dazu muss ein zweiter Kreislauf installiert werden, der das generierte Wissen fremdreferentiell überprüft. Dieser wird durch die Schritte »Verteilung von Wissen«, »Nutzung von Wissen« und »Revision des Wissens« erreicht. So wird eine sog. »doppelte Wissensbuchführung« eingeführt, die sicherstellt, dass nicht nur nach dem »Nutzen«, sondern auch nach den »Kosten« des Wissens gefragt wird.[43] Leitfrage der Revision ist, welches Wissen wozu gebraucht wird und welchen organisationalen Mehrwert es bringt.[44]

Da dieser Kreislauf als Grundlage der vorliegenden Arbeit dient, wird er kurz skizziert:

- **Generierung von Wissen:** Erforderliches Wissen muss generiert werden, entweder ist dieses Wissen schon vorhanden oder es muss fremdes Wissen übernommen werden.[45]
- **Topographien von Wissen:** Auf der organisationalen Ebene geht es darum, Wissen und das komplementäre Nichtwissen festzustellen und eine Be-

[43]vgl. Willke 2001, S. 86 ff.
[44]vgl. ebd., S. 87.
[45]Willke nennt die Übernahme fremden Wissens »SIS – steal ideas shamelessly«, (vgl. Willke 2007, S. 109)

standsaufnahme zu machen.[46] Auf persönlicher Ebene bedeutet Topographie, dass Individuen ihr Wissen so explizieren müssen, dass es in einen sinnvollen Kontext eingeordnet werden kann, sog. »sensemaking«[47].

- **Dokumentation von Wissen:** Relevantes Wissen muss dokumentiert bzw. expliziert und aufbereitet werden, so dass es genutzt werden kann.[48]
- **Verteilung / Nutzung von Wissen:** Hier geht es darum, einen möglichst breiten Zugang zum dokumentierten Wissen zu geben und Nutzungsbarrieren (»übertriebene Abschottung, überflüssige Geheimniskrämerei oder gezielte Filterung von Informationen und ihre Nutzung als Herrschaftswissen«) abzubauen.[49]
- **Revision des Wissens:** Erst wenn regelmäßig geprüft wird, ob das Wissen noch strategisch relevant ist, ist der Kreislauf des Wissensmanagement als Geschäftsprozess geschlossen. Die Revision muss organisiert werden, z. B. durch organisiertes Entlernen, Verlernen oder Vergessen. Willke weist darauf hin, dass der Prozess des Entlernens für Individuen besonders schmerzhaft sein kann, da es eine seine Identität konstituierende Ressource darstellt. Daher müssen gleichzeitig zum Entlernen neue attraktive Alternativen aufgezeigt werden.[50]

Willke nennt mit dem sog. Mikroartikel ein Beispiel für diesen Kreislauf, dabei handelt es sich um eine Art Vorlage zur Dokumentation von Erfahrungsberichten.[51]

Das Grazer Metamodell von Schneider (Abbildung 3) ordnet bisherige Wissensmanagementansätze ein. Schneider geht davon aus, dass es nicht eine Grundausrichtung oder einen Ansatz im Wissensmanagement geben könne, sondern sich dieser an Unternehmensgröße, Branche, Funktion, bewusster oder unbewusster Philosophie ausrichtet. Dabei werden »Wissensindustrialisierung« und »Lernende Organisation« als Extreme genannt, welche jeweils ganz unterschiedliche Mittel und Wege zur Zielerreichung haben.[52]

Das Modell zeigt drei Dimensionen:

---

[46] vgl. Willke 2007, S. 109.
[47] vgl. ebd., S. 45.
[48] vgl. ebd., S. 110 f.
[49] vgl. ebd., S. 110 f.
[50] vgl. ebd., S. 111 f.
[51] vgl. Willke 2001, S. 107 f.
[52] vgl. Schneider 2001, S. 39.

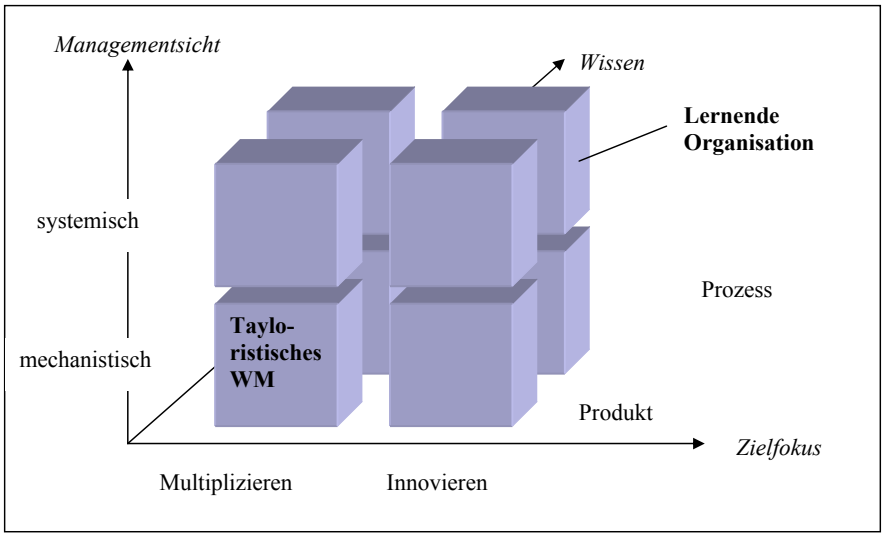

Abbildung 3: Grazer Metamodell des Wissensmanagements (entnommen aus Schneider 2001, S. 32)

- Den Zielfokus von »Multiplizieren« bis »Innovieren« auf der X-Achse,
- die Managementsicht von »mechanistisch« bis »systemisch« auf der Y-Achse,
- und das Verständnis von Wissen auf der Z-Achse, ob es als Produkt oder Prozess gesehen wird.

Damit ergeben sich acht Würfel, die sich zwischen den Extremen des »Tayloristischen Wissensmanagements« und der »Lernenden Organisation« bewegen: Tayloristisches Wissensmanagement zeichnet sich durch standardisierte Darstellung, Speicherung und Zur-Verfügung-Haltung von Wissen aus, was nur elektronisch zu bewältigen ist.[53] Das Gegenextrem bildet die Lernende Organisation, die Wissen als Erkenntnis- und Kommunikationsprozess sieht. Zentral ist die indirekte Kontextsteuerung durch Visionen, Werte und Ressourcen - IT ist eine notwendige, aber keine hinreichende Bedingung.[54]

Tabelle 1 zeigt die Unterschiede zwischen den Extremen des dokumenten- und des personenorientierten Ansatzes im Wissensmanagement.

[53]vgl. Schneider 2001, S. 35 f.
[54]vgl. ebd., S. 37 f.

Tabelle 1: Personen- und Dokumentenorientierte Zugänge (entnommen aus Schneider 2001, S. 89)

Dokumentenorientierter Zugang	Personenorientierter Zugang
Fokus	
Explizieren, Kodifizieren	Kommunizieren, Kooperieren
Aktivitäten	
Identifizieren, Extrahieren, Strukturieren, Visualisieren, Speichern, Warten	Verbinden, Vernetzen, Entwickeln
Werkzeuge	
Intranets, Extranets, Datenbanken, Suchmaschinen, Data-Minings-Programme, Agenten, Filter usw.	Meetings, informelle Treffen, Mentorenprogramme, Personalentwicklung und Teamentwicklung
Probleme	
Wissenstrukturierung (Thesaurus), Sprachauswahl, Filterung, Automatisches Matching	Kultur des Teilens und der Zusammenarbeit, Verstehen, Vermeiden interkultureller Missverständnisse, Entlastung der Mentorenstars
Eignung für	
- Strategien der Kostenführerschaft in reifen Branchen - wohlstrukturierte, eindeutige Inhalte	- Strategien der Innovationen und Kundennähe in Pionierbranchen - schlecht strukturierte, mehrdeutige Inhalte

Im englischsprachigen Raum ist das Modell von Earl 2001 verbreitet. Es wird hier genannt, da Literatur zu Wissensmanagement, agilen Methoden und Software Engineering fast ausschließlich englischsprachig verfasst ist und sich die Autoren auf das Modell von Earl beziehen, wie z. B. Schneider 2009[55], Babar u. a. 2009 und Bjørnson und Dingsøyr 2008, die auch in der vorliegenden Arbeit behandelt werden.

Auch Earl klassifiziert Wissensmanagementansätze in einem Metamodell, in sog. »schools«. Dazu wird eine Grobeinteilung zwischen technokratischer

[55]Hinweis: Bei Schneider 2009 und Schneider 2001 besteht Verwechslungsgefahr.

(»technocratic«), ökomonischer (»economic«) und verhaltensorientierter (»behavioral«) Schule vorgenommen. Die technokratischen Schulen unterteilen sich in »systems«, welche sich auf Technologie zur Wissensspeicherung und -verteilung beruft, sowie in die »cartographic« Schule, welche Landkarten und Verzeichnisse (»yellow pages«, »who-knows-what«) zur Wissenidentifikation verwendet. Die »engineering« Schule fokussiert sich auf Wissensflüsse innerhalb von Geschäftsprozessen, während die »economic« School betrachtet, wie das Unternehmenswissen direkt Einkommen erzeugen kann, z. B. in Beratungen. Die »behavioral« Schule besteht aus drei Unterteilungen: Die »organizational« Schule bezieht sich auf die Pflege von Wissensnetzwerken, die »spatial« Schule besagt, dass die Büroaufteilung möglichst zur Wissensteilung anregen soll, zuletzt die »strategic« Schule, welche Wissen als essentiellen Bestandteil der gesamten strategischen Ausrichtung eines Unternehmens sieht.[56]

Tabelle 2: Earl's schools of knowledge management (entnommen aus Bjørnson und Dingsøyr 2008, S. 3)

	Technocratic			**Economic commercial**	**Behavioural**		
	Systems	**Cartographic**	**Engineering**		**Organizational**	**Spatial**	**Strategic**
Focus	Technology	Maps	Processes	Income	Networks	Space	Mindset
Aim	Knowledge bases	Knowledge Directories	Knowledge flows	Knowledge assets	Knowledge pooling	Knowledge exchange	Knowledge capabilities
Unit	Domain	Enterprise	Activity	Know-how	Communities	Place	Business

Probst u. a. ordnen Wissensmanagement und seine Wissensziele in Anlehnung an das St. Galler Managementmodell in die drei Ebenen normativ, strategisch und operativ ein.[57] Zur klaren Abgrenzung der Wissensmanagementebenen soll dieses Konzept später bei der Modellentwicklung als weitere Grundlage dienen und wird daher kurz vorgestellt:

- Normative Wissensziele spiegeln die politischen und kulturellen »Leitplanken«[58] wider. Sie sollen sich aus der Vision und grundlegenden Ausrichtung des Unternehmens, z. B. tayloristische vs. systemische Managementorientierung, ableiten.[59]

[56] vgl. Bjørnson und Dingsøyr 2008, S. 3.
[57] vgl. Probst u. a. 2010, S. 40 f.
[58] vgl. ebd., S. 41.
[59] Orientierung kann z. B. das vorgestellte Grazer Metamodell geben.

- Strategische Wissensziele sind längerfristige Maßnahmen, um die normativen Wissensziele zu erreichen.
- Operative Wissensziele unterstützen die strategischen Programme, so dass diese auch wirksam umgesetzt werden können.

2.2 Software Engineering und seine Wissensbereiche

In diesem Abschnitt werden kurz die »Wissensbereiche von Software Engineering« vorgestellt, um einen Überblick dieser Disziplin zu bekommen. Danach werden für diese Arbeit relevante Wissensformen bzw. -arten entwickelt. Zum Abschluss werden die Hauptunterschiede traditioneller und agiler Softwareentwicklung gegenübergestellt und das Agile Manifest vorgestellt.

2.2.1 Wissensbereiche von Software Engineering

Der Begriff »Software Engineering« wurde bereits im Jahre 1968 eingeführt. Softwareentwicklung sollte damit zu einer Ingeneursdiziplin werden.[60]

Eine Systematisierung in die Aktivitäten und Wissensbereiche liefert das Werk SWEBOK (Guide to the Software Engineering Body of Knowledge), welches vom IEEE (Institute of Electrical and Electronics Engineers) herausgegeben und gepflegt wird.[61] SWEBOK versteht sich als Referenzwerk für das Software Engineering. Um einen kurzen Überblick zu geben, werden hier knapp die einzelnen Wissensbereiche beschrieben:

- **Software requirements** bezieht sich auf die Anforderungen, die an eine Software gestellt werden. Dabei geht es auch um den Prozess der »Extraktion«, Analyse, Validierung (z. B. durch Akzeptanztests) und Spezifikation der Anforderungen.
- **Software design** bezeichnet den Prozess der Definition von Architektur, Komponenten und anderer Charakteristika eines Softwaresystems. Es bezeichnet somit auf der Makroebene das Grundgerüst der Software oder Komponenten, aber auch die Benutzung von Design patterns oder Prinzipien wie »information hiding« auf der Mikroebene.

[60] vgl. Schneider 2009, S. 46.
[61] Abran u. a. 2004.

- **Software construction** bezeichnet das eigentliche Programmieren inkl. Validierung z. B. durch Unit tests, aber auch verwendete Programmiersprachen.
- Im **Software testing** geht es um den Prozess, verwendete Techniken/ Programme, das Aufstellen und die Durchführung von Testplänen sowie die Unterscheidung unterschiedlicher Testlevel (z. B. Unit-Tests, Akzeptanztests oder auch Performance- und Stresstests).
- **Software maintenance** beschreibt neben der Notwendigkeit von Softwarepflege bzw. Weiterentwicklung auch den Prozess dahinter.
- Im **Software configuration management** wird das »Ausrollen« bzw. »Deployment« oder »Release« von Software beschrieben. Es hängt besonders davon ab, in welchem Umfeld die Software eingesetzt werden soll: Wird sie direkt mit einer Hardware gebündelt oder ist es eine Webanwendung, die auf einem Webserver ausgeführt wird? Im Folgenden soll der Bereich auf Webanwendungen im Sinne von Allspaw und Robbins 2010 und Humble und Farley 2010 beschränkt werden und den Betrieb der (Web-)Software einschließen.
- **Software engineering management** bezieht sich auf Managementaktivtäten im Rahmen von Softwareprojekten, z. B. die Planung des Prozesses, Qualitäts- und Risikomanagement.
- **Software engineering process** beinhaltet die Implementierung, Definition sowie Messtechniken für den Softwarentwicklungsprozess oder das Produkt an sich.
- **Software engineering tools and methods** beinhaltet als Unterbereiche noch einmal alle Wissensbereiche und beleuchtet Werkzeuge, die jeweils unterstützend wirken können, z. B. Issue tracker.
- **Software quality** bezieht sich auf allgemeine Bereiche wie Softwareethik (Qualitätsanspruch), aber auch konkret, wie Qualitätssicherung funktioniert und welche Methoden es dafür gibt.

Erwähnenswert ist, dass sich mit dem »Web Engineering« eine spezielle Untergruppe des Software Engineering herausgebildet hat.[62] Dieser Bereich beleuchtet die speziellen Aspekte von Webanwendungen wie z. B. verwendete Protokolle und Besonderheiten von Anwendungen, die in Webbrowsern laufen. Eine Kategorisierung aus dem Blickwinkel von SWEBOK hat Navarro 2009

[62] vgl. Navarro 2009.

vorgenommen und festgestellt, dass SWEBOK größtenteils auf Webanwendungen anwendbar ist.

2.2.2 Arten von Wissen

Im Software Engineering kommen mehrere Wissensarten vor. D. h. neben implizitem und explizitem Wissen kann noch eine andere Unterteilung vorgenommen bzw. eine weitere Dimension definiert werden.

Rus und Lindvall 2002 besagen: »Software engineering involves several knowledge types - technical, managerial, domain, corporate, product, and project knowledge.« Auch Babar u. a. 2009 nehmen eine Unterteilung von Wissensarten vor. Neben implizit/ explizit fügen sie die Dimension applikationsspezifisches/ applikationsgenerisches Wissen hinzu. Damit ist gemeint, dass es Wissen gibt, welches speziell eine Applikation betrifft, und darüber hinaus generisches Wissen, welches unabhängig von einem Projekt bzw. einer Applikation (wieder-)verwendet werden kann.

Aus Rus und Lindvall 2002, Babar u. a. 2009 und SWEBOK lassen sich grob die in Tabelle 3 gezeigten Wissensarten ableiten. Vorgeschlagen wird eine Unterteilung in Domänenwissen, Anforderungswissen, Applikationswissen, Prozesswissen. Es sind jeweils Synonyme aufgezählt und Beispiele zur Veranschaulichung gegeben. Diese Wissensarten sollen neben der Dimension »implizit/ explizit« als weitere Dimension und in dieser Arbeit als grundlegende Einteilung gesehen werden.

Tabelle 3: Wissensarten im Software Engineering

Wissensart	Synonyme	Beschreibung	Beispiel(e)
Domänenwissen	Domain knowledge[63], Business knowledge	Wissensdomäne der Software	- Bei einer Steuersoftware Wissen über das Steuerrecht.
Anforderungswissen	Produktwissen, Requirements	Wissen über Anforderungen sowie Nutzungsverhalten	- Welche Anforderungen werden an die Software gestellt? - Sind die Anforderungen noch aktuell und marktgerecht? - Wie ist das Nutzungsverhalten? Ob und wie werden Features benutzt?
Applikationswissen	application-specific knowledge, technical knowledge	Applikations-spezifisches Detailwissen	- Aufbau und Struktur der Software und seiner Komponenten[64] - Verwendete Architekturansätze, z. B. SOA (Service oriented architecture)[65]
Prozesswissen	Projektwissen, corporate knowledge, Legal knowledge	Wissen über den Softwareentwicklungsprozess und das Projektumfeld	- Welche Aufgabe soll als nächstes erledigt werden? - Wer sind die Projektbeteiligten? - Wann ist eine Aufgabe fertig? - Wie sind die Schritte, um ein neues Software Release zu erstellen - Rechtliche Rahmenbedingungen
Erfahrungswissen	Handlungswissen, Fachwissen, application-generic knowledge	Generisches Erfahrungswissen	- Beherrschen mehrerer Programmiersprachen - Muster, Werkzeuge und Methoden[66] - Referenzarchitekturen[67]

[63] vgl. Abran u. a. 2004, 2–5.
[64] vgl. ebd., 6–4.
[65] vgl. Babar u. a. 2009, S. 26.
[66] vgl. Abran u. a. 2004, 10–1.
[67] vgl. Babar u. a. 2009, S. 26.

2.2.3 Unterschiede traditioneller und agiler Softwareentwicklung

Im Software Engineering haben sich zwei Strömungen herausgebildet: Traditionelle Softwareentwicklung und agile Softwareentwicklung. Tabelle 4 zeigt im Überblick die Hauptunterschiede anhand der Kriterien Grundannahme, Managementstil, Wissensmanagement, Kommunikation, Entwicklungsmodell, gewünschte organisationale Form/ Struktur und die Art der Qualitätskontrolle.

Tabelle 4: Hauptunterschiede zwischen traditioneller und agiler Softwareentwicklung (entnommen und übersetzt aus Dybå und Dingsøyr 2008, S. 4)

	Traditionelle Entwicklung	**Agile Entwicklung**
Grundannahme	Systeme sind vollkommen spezifizierbar, vorhersagbar, und werden durch extensives und sorgfältiges Planen erstellt	Hochqualitative, anpassbare Software wird durch kleine Teams entwickelt, die den Prinzipien (1) kontinuierlicher (Software-) Designverbesserung und (2) des Testens basierend auf schneller Rückkopplung (Feedback) und Veränderung folgen
Managementstil	Direkte Steuerung (»Command and control«)	Führerschaft (»Leadership«) und Zusammenarbeit
Wissensmanagement	explizit	implizit
Kommunikation	formal	informell
Entwicklungsmodell	Lebenszyklus-Modell (Wasserfall, Spiralmodell und Variationen)	Modell der evolutionären Entwicklung und Auslieferung
Gewünschte organisationale Form/ Struktur	Mechanistisch (bürokratisch mit großer Formalisierung), angelegt auf große Organisationen	Organisch (flexibel und partizipativ, bestärkt gegenseitige soziale Interaktion), geeignet für kleine und mittlere Unternehmen
Qualitätskontrolle	Erhebliche Planung und strikte Steuerung. Spätes, intensives Testen	Kontinuierliche Kontrolle/ Steuerung von Anforderungen, (Software-)Design und Lösungen. Kontinuierliches Testen.

Zur agilen Softwareentwicklung: 2001 wurde das »Agile Manifest« von einer Gruppe erfahrener Softwareentwicklern als Fortentwicklung des Lean Software Development formuliert. Es enthält vier Kernwerte[68]:

> Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:
>
> - Individuen und Interaktionen mehr als Prozesse und Werkzeuge
> - Funktionierende Software mehr als umfassende Dokumentation
> - Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
> - Reagieren auf Veränderung mehr als das Befolgen eines Plans
>
> Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.

Zur Erläuterung der vier Kernwerte werden zwölf Prinzipien genannt:[69]:

> - Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.
> - Heisse Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
> - Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
> - Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
> - Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
> - Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteam zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
> - Funktionierende Software ist das wichtigste Fortschrittsmaß.
> - Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
> - Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
> - Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.
> - Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.
> - In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Nachdem Software Engineering in seinen Wissensbereichen, -arten und Herangehensweisen kurz beleuchtet wurde, wird nun der Blick auf den aktuellen Stand des Wissensmanagement im Software Engineering gegeben werden.

---

[68] Die folgenden Ausführungen sind direkte Zitate von URL: `http://agilemanifesto.org/iso/de/` (abgerufen am 15.11.2012)

[69] Die folgenden Ausführungen sind direkte Zitate von URL: `http://agilemanifesto.org/iso/de/principles.html` (abgerufen am 15.11.2012)

2.3 Überblick bisheriger Forschungsansätze

Im Folgenden werden bisherige Ansätze, die sich explizit mit Wissensmanagement im Software Engineering beschäftigen, kurz vorgestellt. Die Literatur zu diesem Thema ist relativ überschaubar. Lediglich Schneider 2009, Babar u. a. 2009 sowie einige Aufsätze (Chau u. a. 2003, Melnik und Maurer 2004, Levy und Hazzan 2009) sowie eine Studie von Bjørnson und Dingsøyr 2008 lagen vor.

Schneider 2009 unterscheidet zwischen »Knowledge« (Faktenwissen) und »Experience« (Erfahrungswissen).[70] Zur Strukturierung von Faktenwissen stellt er grundlegende Werkzeuge wie Use Cases, Glossare und Domain models sowie UML vor und geht auf Ontologien (Semantic Web) ein. Basierend auf dem Konzept der »Experience Factory« von Basili u. a. 1994 entwickelt Schneider das »Experience management« Konzept. Es handelt sich um ein Verfahren, mit dem Erfahrungsberichte generiert, aufbereitet, verteilt und genutzt werden und welches als SEC (Software Experience Center) bei Daimler Chrysler im Einsatz ist[71].

Schneider weist auf Hindernisse und Probleme bei der Anfüllung und Nutzung von solchen Repositories (Wissensspeicher) hin, z. B. soll nicht davon ausgegangen werden, dass nur durch Aufstellen einer technischen Lösung Experten eingeständig damit beginnen, ihre Erfahrungsberichte dort einzustellen.[72] Weiterhin gibt es Probleme bei der Nutzung, bspw. durch fehlendes Vertrauen und das »Not-invented-here« Syndrom.[73]

Ergänzend führt Schneider Techniken zur Aktivierung bzw. Explizierung von Wissen auf wie Interviews, post mortems und Workshops sowie Hilfestellungen bei der Verteilung, z. B. durch case-based Reasoning.[74] Weiterhin nennt er Konzepte wie »Experience Broker«, eine Art Anlaufstelle, die zur Aufgabe hat, Wissensproduzenten und -bedürfnisse zu finden verknüpfen, z. B. über Gespräche mit Personen in Kaffeeecken.[75] Eine technische Lösung benennt Schneider mit den Expertennetzwerken, die zur Identifizierung von internen

[70] vgl. Schneider 2009, S. vii f.
[71] Tatsächlich ist SEC ein Zusammenschluss von mehreren Großunternehmen (ABB, DaimlerChrysler, Motorola, und Nokia), vgl. Lindvall u. a. 2004.
[72] vgl. Schneider 2009, S. 157 f.
[73] vgl. ebd., S. 159 f.
[74] vgl. ebd., S. 174.
[75] vgl. ebd., S. 170 f.

Experten dienen, um so Verknüpfungen herstellen zu können.[76] Agile Vorgehensmodelle werden bei Schneider als Trends, welche kommen und gehen, bezeichnet und nicht weiter aufgegriffen.[77]

Schneider 2009 bietet einen Überblick über Wissensmanagement im Software Engineering, beschränkt sich aber sehr auf die Vorstellung seines eigenen Ansatzes, so wird beispielsweise eine Ausrichtung des Wissensmanagementansatzes an der Unternehmensstrategie nicht diskutiert.

Babar u. a. ziehen ihren Fokus enger und beschäftigen sich mit Wissensmanagement in der Software Architektur. Die Autoren beschreiben größtenteils Wissensmanagementwerkzeuge und liefern einige Erfahrungberichte, gehen aber nur randständig auf agile Methoden ein. Sie beziehen sich auf das Modell von Earl, agiles Vorgehen wird von ihnen in der »spatial school« verortet. Damit ist gemeint, dass Wissensmanagement durch Raumaufteilung und Kommunikation erfolgt. Sie sagen, dass agile Softwareentwicklungsteams die gleichen Methoden zur Wissensverteilung und -bewahrung benutzen können, die sie bereits einsetzen, z. B. Boards zur Kommunikation. Dies könnte für agile Teams ein ausreichendes Wissensmanagement darstellen.[78]

Erwähnenswert ist noch, dass Babar u. a. in Anlehnung an das Bausteinmodell von Probst u. a. 2010, welches einen vollständigen Managementkreislauf abbildet, eine Abwandlung dessen vorstellen: Es wird z. B. vorgeschlagen, quantifizierbare Ziele aufzustellen, beispielsweise »number and positions of employees who accessed or contributed to organizational knowledge repository«, welche dann später in der »Wissensbewertung« wieder ausgewertet werden können.[79] D. h. hier wird das Wissensmanagement im Gegensatz zu Schneider 2009 ansatzweise als Managementaktivität in den Unternehmenskontext eingebunden.

Neben Fallstudien werden noch allgemeine Methoden zur Unterstützung wie z. B. CoP (Community of Practice), Web 2.0 sowie Wikis aufgeführt.[80]

Insgesamt betrachtet geben Babar u. a. 2009 neben generellen Konzepten einen guten Einblick in den Bereich Softwarearchitektur und Wissensmanage-

[76] vgl. Schneider 2009, S. 176.
[77] vgl. ebd., S. 199.
[78] vgl. Babar u. a. 2009, S. 68.
[79] vgl. ebd., S. 82 ff.
[80] vgl. ebd., S. 126.

ment. Dabei weisen sie auf spezielle Werkzeuge hin, die dazu dienen, Wissen über Softwarearchitekturen abzusichern.

Bjørnson und Dingsøyr 2008 untersuchen in ihrer Studie Veröffentlichungen zu Forschungsansätzen, die sich mit Software Engineering und Wissensmanagement beschäftigen. Für die systematische Betrachtung des Feldes legen sie das Modell von Earl zugrunde. Demnach fokussiert sich der Großteil der Studien auf technokratische sowie verhaltensorientierte und nur einige wenige auf ökonomische, räumliche oder kartografische Aspekte. In erster Linie beschäftigten sich Untersuchungen zum zum Wissensmanagement im Software Engineering bisher mit der Speicherung und Abfrage von Wissen, wohingegen die Gewinnung, der Transfer und die Anwendung von Wissen noch wenig beachtet wurden.

Bjørnson und Dingsøyr gehen der Frage nach, welche Relevanz die »schools« nach Earl für das Software Engineering haben. Zunächst stellen sie fest, dass zwischen zwei Vorgehensweisen in Unternehmen unterschieden werden kann, die grundlegende Auswirkungen für das Wissensmanagement haben: Entweder werden agile Entwicklungsmethoden verfolgt, dann beziehen sich die Aktivitäten des Wissensmanagements auf implizites Wissen; oder Unternehmen arbeiten mit traditionellen Entwicklungsmethoden, in diesem Fall steht explizites Wissen im Zentrum der Wissensmanagementaktivitäten.

Die meisten Veröffentlichungen sind zu technokratischen Ansätzen (»technocratic school«) zu finden, sie blicken auf die Entwicklung von Werkzeugen im Software Engineering und betrachten die Entwicklung und den Gebrauch von Wissenspeichern. Dabei beschäftigen sich diese Untersuchungen entweder mit spezifischen Aktivitäten oder mit dem gesamten Entwicklungsprozess. Im ersten Fall kristallisieren sich vier Hauptbereiche heraus: Formale Routinen, Mapping von Wissensflüssen, Bewertungen von Projekten, soziale Interaktion. Im zweiten Fall wird aufgezeigt, dass bestehende Entwicklungsprozesse Kommunikation sowie Lernen verbessern. Die Autoren weisen darauf hin, dass das Teilen von implizitem Wissen wichtig ist, um die Praxis weiterzuentwickeln und in der Forschung mehr Beachtung finden sollte.

In den Bereichen »systems« und »engineering« ist festzustellen, dass eine technische Infrastruktur bestehen muss, um Wissensteilung überhaupt ermöglichen zu können, wobei das Problem der tatsächlichen Nutzung hinzukommt. Bjørnson und Dingsøyr weisen darauf hin, dass zwar implizites Wissen wohl nicht bedeutungsvoller für die Softwareentwicklung wäre, als explizites Wis-

sen, Forschungsergebnisse zu agiler Entwicklung aber zeigen, dass es möglich ist, qualitativ hochwertige Software zu entwickeln, ohne ein aufwändiges Management von explizitem Wissen zu betreiben.

Neben den Studien mit Blick auf technokratische Ansätze (»technocratic school«), sind Veröffentlichungen zu finden, die sich auf Verhaltensaspekte fokussieren. Diese beschäftigen sich mit organisationalen und strategischen Aspekten und behandeln den Nutzen der Vernetzung zwischen Personen. Es wurden keine Studien ausgemacht, die sich auf räumliche Aspekte (»cartographic school«) blicken.

Bjørnson und Dingsøyr 2008 kommen zu dem Schluss, dass die zukünftige Forschungspraxis sich auf die »organisational school«, die sich mit dem Management von implizitem Wissen befasst, fokussieren sollte, und nicht ein ausschließlicher Fokus auf explizitem Wissen und dessen Kodifizierung liegen soll. Agile Softwareentwicklung ist ein Trend, der in Zukunft die industrielle Praxis beeinflussen wird.[81]

Melnik und Maurer 2004 überprüfen anhand eines Experiments, wie effektiv Wissensverteilung durch Kodifizierung oder durch direkte Kommunikation ist. Es wird festgestellt, dass während tayloristische Ansätze Wissen als Objekt behandeln, z. B. in Form von Dokumentationen, Wissensspeichern (»knowledge bases«), agile Ansätze Wissen als Beziehung, also als kommunikativen Prozess, behandeln. Dennoch wird Dokumentation auch in agilen Ansätzen nicht prinzipiell abgelehnt. Die Autoren kommen zu dem Schluss, dass mit steigender Komplexität auch die Notwendigkeit für direkte Kommunikation steigt. Gerade in Situationen mit hoher Unsicherheit und Wahrscheinlichkeit von Missverständnissen, wie der Anforderungsanalyse von Software, sind direkte Verbindungen (»face-to-face«) wichtig.

Chau u. a. 2003 stellen »tayloristische« und »agile« Wissensteilung gegenüber. Sie untersuchen die unterschiedlichen Ansätze anhand der Aspekte Dokumentation, Anforderungs- und Domänenwissen, Training, Kompetenzmanagement, Vertrauen und Fürsorge, Teamzusammenstellung, kontinuierliches Lernen und Wissenspeichern.

Insgesamt kommen sie zu dem Ergebnis, dass tayloristische Ansätze sich beim Wissensaustausch auf Dokumente und Wissensspeicher wie »Experi-

[81] vgl. Bjørnson und Dingsøyr 2008, S. 11 f.

ence Factory« stützen, während agile Vorgehensweisen stark auf direkter Kommunikation und kontinuierlichem Lernen durch schnelle Rückkopplung mittels inkrementeller Entwicklung basieren.

Hauptkritik an Wissensspeichern ist, dass es nicht klar ist, inwiefern gespeicherte Erfahrungsberichte auch wieder internalisiert und damit genutzt werden; auch ist der Aufbau und die Pflege mit Kosten verbunden. Allerdings können zentral gepflegte Speicher organisationales Lernen unterstützen. Agile Ansätze fördern durch die vielen Methoden der direkten Kommunikation einen effektiven Wissensaustausch innerhalb der Teams; Wissenstransfer auf organisationaler Ebene leisten agile Ansätze allerdings nicht, so die Autoren. Als technologische Lösung für die Teilung von Erfahrungsberichten in agilen Teams wird ein Wiki vorgeschlagen, welches kollaborativ gepflegt wird.[82]

Levy und Hazzan 2009 benutzen erstmals den Begriff »Agile Knowledge Management«. Sie vermuten, dass agile Methoden implizit Wissensmanagement fördern, nehmen aber keine weitere Systematisierung vor. Es wird ein weiterer Forschungsbedarf konstatiert, z. B. mittels Beobachtungen und Interviews. Nach Levy und Hazzan ist die Paarung von Wissensmanagement und agilen Methoden nicht neu, und auch nicht überraschend in Anbetracht dessen, dass beide Ansätze mit Unternehmenskultur und »change management« zu tun haben.[83]

Zusammengefasst lässt sich über den Forschungsstand zu Wissensmanagement im Software Engineering, und speziell im agilen Umfeld, also sagen:

- Die Literatur nimmt bereits eine Einteilung zwischen den Extremen personenorientiertes (»verbal«, »communication«, »personalization«, »socialization«, »tacit knowledge«, »agile«[84]) und dokumentenorientiertes (»codification«, »externalisation«, »capture«, »explicit knowledge«, »repository«, »technocratic«, »documented«[85]) Wissensmanagement vor.
- Agile Softwareentwicklung kann also im personenorientierten Wissensmanagement von Schneider 2001 (vgl. Tabelle 1) verortet werden.

[82] vgl. Chau u. a. 2003, S. 307.
[83] vgl. Levy und Hazzan 2009, S. 64.
[84] vgl. Bjørnson und Dingsøyr 2008, S. 4; vgl. Earl 2001, S. 225; vgl. Chau u. a. 2003, S. 302; vgl. Melnik und Maurer 2004, S. 21; vgl. Babar u. a. 2009, S. 60.
[85] vgl. Bjørnson und Dingsøyr 2008, S. 6; vgl. Chau u. a. 2003, S. 306; vgl. Melnik und Maurer 2004, S. 21; vgl. Schneider 2009, S. 179; vgl. Earl 2001, S. 220.

- Die Einteilung agiler Softwareentwicklung in die Taxonomien von Earl fällt in die »spatial school«(Raumaufteilung), teilweise auch in die »engineering school« da Wissen innerhalb des (agilen) Softwareentwicklungsprozesses ausgetauscht wird.[86]
- Levy und Hazzan 2009 stellen fest, dass agile Methoden und Wissensmanagement grundlegend verwandt sind. Für sie besteht Forschungsbedarf in Bezug darauf, inwiefern agile Methoden Wissensmanagement unterstützen oder leisten können.

2.4 Kurzvorstellung agiler Softwareentwicklungssmodelle

In diesem Abschnitt werden agile sowie Lean Vorgehensmodelle kurz vorgestellt, beginnend mit Scrum sowie Extreme Programming. Danach werden als neuere Modelle Software Kanban und DevOps miteinbezogen. Sie sind einerseits neuer und nehmen auch Lean Prinzipien auf. Anderseits betrachten sie neben der Softwareentwicklung auch den Betrieb der Software (Operations[87]) und lassen somit die Entwicklung eines vollständigen Bildes zu.

Scrum, XP, Software Kanban und DevOps sollen als Modelle verstanden werden, während die einzelnen Bausteine der Modelle als Methoden bezeichnet werden.

2.4.1 Scrum

Scrum ist ein Methodenframework zur Steuerung von Softwareprojekten, Produkt- und Softwareentwicklung. Es setzt dabei auf inkrementelles Vorgehen im Entwicklungsprozess, um z. B. besser auf unvorhergesehene Ereignisse oder Prioritätsveränderungen reagieren zu können. Scrum-Teams sind selbstgesteuert und crossfunktional.[88]

Die Ursprünge von Scrum sind interessanterweise auch die gleichen des japanischen Wissensmanagements: Erstmals stellten Nonaka und Takeuchi 1986 einen neuen Ansatz zur Produktentwicklung vor. Diesen erweiterten Nonaka

[86] vgl. Bjørnson und Dingsøyr 2008, S. 7; vgl. Babar u. a. 2009, S. 67.
[87] »Operations« bezeichnet hier den Betrieb der Websoftware, also z. B. Auslieferung der Anwendung durch einen Webserver, vgl. Navarro 2009, S. 3178.
[88] vgl. Dybå und Dingsøyr 2008, S. 3.

und Takeuchi 1995, dort heißt er noch »Rugby approach« und bezieht sich auf crossfunktionale Teams und überlappende Entwicklungszyklen.[89]

Tabelle 5 zeigt eine Übersicht der Techniken hinter Scrum[90].

Tabelle 5: Methoden von Scrum

Methode	Beschreibung
Product Backlog	Liste von allen zu implementieren Anforderungen. Verantwortlich für das Pflegen und Priorisieren ist der »Product Owner« (Rolle).
Effort Estimation	Aufwandsschätzung von Anforderungen aus dem Product Backlog.
Sprint	Ein Entwicklungszyklus, um ein neues Produktinkrement zu erstellen. Er beträgt normalerweise zwei bis vier Wochen.
Daily Meeting	Tägliches, kurzes Teammeeting zur Abstimmung des Fortschritts und zur Risiko- bzw. Problemanalyse. Wird auch synonym »Standup Meeting« genannt.
Sprint Planning Meeting	Planungsmeeting für den nächsten Sprint. Aus dem Backlog werden Anforderungen (bzw. »User stories«) ausgewählt und kleinere Aufgaben aufgespaltet.
Sprint Backlog	Liste der noch zu erledigenden User stories für einen Sprint / Iteration
Sprint Review Meeting	Treffen des Teams zum Ende eines Sprints zusammen mit allen Stakeholdern zur Demonstration des fertiggestellten Produktinkrements.
Sprint Retrospective	Regelmäßiges Treffen des Teams, um den letzten Sprint zu reflektieren und Verbessungspotenziale aufzudecken.
Sprint Burn Down Chart	Visuelle Darstellung der noch zu erledigen Arbeit / User stories im Sprint.
selbstorganisierte, crossfunktionale Teams[91]	Scrum- bzw. agile Teams bestehen zumeist aus selbstorganisierten, crossfunktionalen Teams, d. h. alle nötigen Funktionen und Fachkompetenzen zur Produktentwicklung sind im Team vorhanden.[92]

2.4.2 Extreme Programming

XP (Extreme Programming) ist eine Softwareentwicklungsmethode, spezialisiert auf Softwarequalität, Kundennähe und Anpassungsfähigkeit an sich schnell ändernde Softwareanforderungen.[93] Im Gegensatz zu Scrum steht hier die Softwareentwicklung und nicht die Projektsteuerung im Vordergrund.[94]

Tabelle 6 zeigt eine Übersicht der Techniken hinter XP[95].

[89] vgl. Nonaka und Takeuchi 1995, S. 78 f.
[90] übernommen aus Fernandes und Almeida 2010, S. 393.
[91] vgl. Schwaber 2009, S. 7.
[92] »selbstorganisiert« und »crossfunktional« sollen in dieser Arbeit immer als Einheit gesehen werden, vgl. Nonaka und Takeuchi 1995, S. 76.
[93] vgl. Beck 1999, S. xvii f.
[94] vgl. Dybå und Dingsøyr 2008, S. 3.
[95] übernommen aus Fernandes und Almeida 2010, S. 392.

Tabelle 6: Methoden von XP

Methode	Beschreibung
Planning Game	Planungsmeeting, an dem der Kunde (»Customer«)[96] und das Team teilnehmen, um die nächsten ToDos und Stories zu planen.
Small Releases	Fertig gestellte Stories oder Features sollen so schnell wie möglich in Produktion gehen, d. h. für den Kunden benutzbar sein.
Metaphors	Metaphern sollen dazu dienen, technische Begriffe zu vermeiden, so dass der Kunde und das Team einen gleichen Wortschatz entwickeln.
Simple Design	Die Architektur, der Quelltext und die Tests sollten so einfach wie möglich sein.
TDD (Test-driven Development)	Alle implementierten Eigenschaften sollen durch Tests abgedeckt sein. [97]
Pair Programming	Zwei Programmierer arbeiten simultan an einem Computer. Abwechselnd programmiert einer der beiden (»Driver«), während der andere mitdenkt und den Quelltext laufend auf z. B. Fehler überprüft (»Navigator«).
Collective Code ownership	Jedes Teammitglied darf jede Stelle des Quelltextes verändern, er gehört allen.
Refactoring	Durch Refactoring wird laufend die Qualität des Quelltextes hoch gehalten, z. B. durch Entfernen von dupliziertem Code.
CI (Continuous Integation)	Ist eine Story oder ein in sich abgeschlossener Task fertig, so wird die neue Softwareversion automatisiert validiert (alle Tests werden ausgeführt).
40-Hour weeks	Es soll kein Teammitglied Überstunden machen.
On-site Customer	Der Kunde bzw. Auftraggeber soll Teil des Entwicklungsteams sein, um ein enges Verhältnis zu pflegen.
Coding Standards	Das Team soll sich auf Coding Standards einigen, z. B. welche Programmiersprache oder Design Patterns benutzt werden.

2.4.3 Software Kanban

Software Kanban ist die Übertragung des Kanban-Prozesses, bekannt aus dem Toyota Production System, auf die Softwareentwicklung. Im Gegensatz zu Scrum, welches den Entwicklungsprozess eher vorgibt, adaptiert sich Software Kanban auf einen bestehenden Prozess und ändert diesen evolutionär.[98]

Software Kanban soll durch die Einführung eines »Pull-Prinzips« die Überlastung von Software Teams vermeiden und einen evolutionären Veränderungs- sowie Verbesserungssprozess einleiten. Abbildung 4 zeigt die typischen Elemente von Kanban bzw. eines Kanban Boards: Zuerst wird ein beliebiger beste-

[96] Im Folgenden soll mit »Kunde« derjenige gemeint sein, welcher die Softwareanforderungen liefert. D. h. der Kunde kann auch unternehmensintern sein.
[97] TDD wird später in Test-first Development (TFD) umbenannt und damit präzisiert, d. h. es wird definitiv vor jeder Quelltextänderung erst ein Test geschrieben, der das neue Verhalten beschreibt, vgl. Beck und Andres 2005, S. 31.
[98] vgl. Anderson 2010, S. 63.

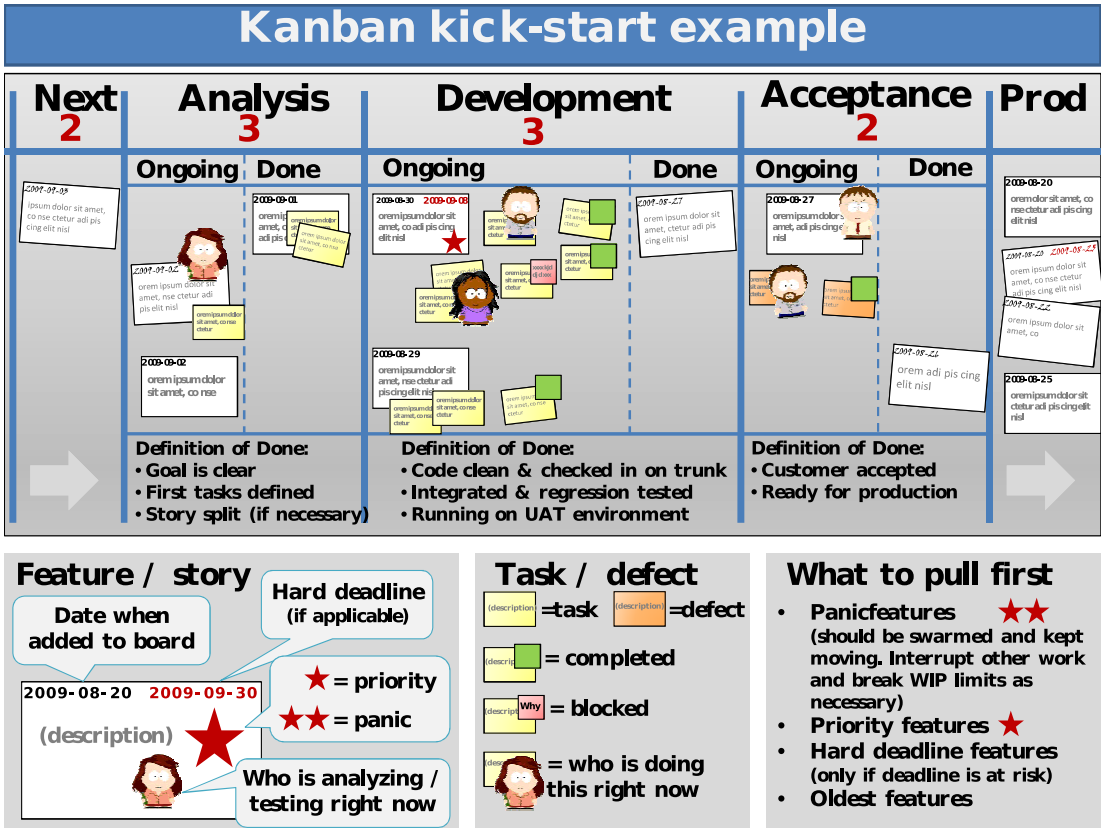

Abbildung 4: Beispiel Kanban Board (entnommen aus URL: `www.crisp.se/file-uploads/kanban-example.pdf` (abgerufen am 15.11.2012))

hender Workflow bzw. die aktuelle Wertschöpfungskette beschrieben und die einzelnen Stationen als Spalten gezeichnet. Jede Aufgabe wird als Karte auf dem Board visualisiert. Sobald eine Station keine Aufgaben mehr zu erledigen hat, »zieht« sie sich eine neue Karte in seine Spalte (von links nach rechts). Hierdurch wird der Fluss bzw. die Wertschöpfungskette (»value stream«) visualisiert.

Durch sogenannte WIP (Work in Progress)-Limits, in der Abbildung als Zahlen zu erkennen, wird für jede Station eine Maximalanzahl von gleichzeitig bearbeitbaren Aufgaben festgelegt. Dies soll den evolutionären und kontinuierlichen Veränderungsprozess (genannt »Kaizen«[99]) auslösen: Engpässe werden visualisiert und können somit aufdeckt werden.

Jedem der Prozessschritte wird dazu noch eine »Definition of Done« (»Explicit Process Policies«) zugeschrieben, welche ausdrückt, wann eine Aufgabe als

[99] vgl. Anderson 2010, S. 40 f.

abgeschlossen gilt.[100] Durch Messungen der Durchlaufzeiten (»Lead time«) oder durch CFD (Cumulative Flow Diagram) kann der Fluss des Gesamtsystems visualisiert, ausgewertet und verbessert werden.[101]

Tabelle 7 zeigt eine Übersicht der Methoden hinter Software Kanban noch einmal im Überblick.[102]

Tabelle 7: Methoden von Software Kanban

Methode	Beschreibung
Visualize Workflow	Visualisierung des Softwareentwicklungsprozesses. Typische Schritte sind: Next, Analyse, Development, Acceptance, Production.
Limit Work-in-Progress	Limitierung der gleichzeitig bearbeitbaren Aufgaben je Station.
Measure and Manage Flow	Messung und Steuerung, z. B. Durchlaufzeiten oder CFD.
Explicit Process Policies	Explizierung von Prozessrichtlinien, also wann eine Aufgabe als fertig gilt, auch »Definition of Done« genannt.
Use Models to Recognize Improvement Opportunities	Durch zusätzliche Modelle wie Theory of Constraints oder Systems thinking sollen weitere Verbesserungsmöglichkeiten für den Prozess aufgezeigt werden.

Der Einfachheit halber werden Kanban und »Software Kanban« im folgenden synonym benutzt.

2.4.4 DevOps

DevOps ist eine relative neue Strömung im Umfeld des Web Engineering. Der Name ist eine Verbindung von »Development« und »Operations«, welches mehrere Gedanken hervorbringt.

Einerseits geht es darum, die Zusammenarbeit von Entwicklungsabteilungen und Operations, also Systemadministratoren, die für den Betrieb zuständig sind, zu verbessern. In der Vergangenheit haben sich dort sog. »Wissenssilos« gebildet, so wissen die Entwickler oft gar nicht, wie sich die von ihnen entwickelte Software in Produktionsumgebungen verhält, bzw. sie bekommen keine Rückmeldung. Auf der anderen Seite sind Administratoren für den reibungslosen Betrieb der Software zuständig und müssen diese überwachen, sie haben aber zumeist gar kein Applikationswissen, sondern folgen mechanistisch den Anweisungen ihrer Dokumentation.[103] Ein prominentes Beispiel ist ein

[100] vgl. ebd., S. 38 f.
[101] vgl. ebd., S. 140 f.
[102] übernommen aus ebd., S. 15.
[103] vgl. Cook u. a. 2012, S. 70 f.

Speicherleck im Programm: Wüsste der Softwareentwickler von einem Speicherleck in der Produktionsumgebung, könnte er es relativ schnell schließen. Der Systemadministrator bemerkt das Speicherleck eher durch seine Überwachungswerkzeuge, hat aber kein Wissen, um es selbstständig zu schließen.

Ein weiterer Trend, der im Namen DevOps steckt, ist die Automatisierung im Web Umfeld. Durch neue technische Entwicklungen wie Cloud (Infrastructure as a Service[104]) und Virtualisierung wird schnellere Beschaffung (»Provisioning«) von Netzwerk- und Rechner-Ressourcen für Anwendungen möglich. Serverfarmen werden programmierbar, damit rückt der Beruf des »Op« hin zu »Dev«.[105]

Da die Definition unklar ist, soll hier das Akronym »CAMS« benutzt werden, welches Willis 2010 eingeführt hat, um DevOps zu umschreiben: Einerseits geht es um die Unternehmenskultur, dann wie erwähnt um Automatisierung, um Metriken und um »Sharing«, d. h. das Teilen zwischen den Abteilungen, z. B. um die erwähnten Wissenssilos zu umgehen.

[104] zum Beispiel Amazon Web Services (URL: `http://aws.amazon.com/` (abgerufen am 15.11.2012))

[105] vgl. ebd., S. 73 f.

Tabelle 7 zeigt eine Übersicht der Prinzipien hinter DevOps[106]

Tabelle 8: Prinzipien von DevOps

Prinzip	Beschreibung	Methoden
Culture	Auflösung von Wissenssilos wie »Development« und »Operations«, Systemdenken (»systems thinking«)	**Automation, Measurement und Sharing** sind Teil der DevOps-Kultur, d. h. eine **offene teilungsbereite und fehlertolerante Unternehmenskultur** ist Vorraussetung für DevOps[107]
Automation	Automatisierung von wiederkehrenden Aufgaben (z. B. Deployment), Reproduzierbarkeit von Testumgebungen für Releases.[108]	- Continuous Delivery[109] - Infrastructure as Code[110]
Measurement	Messung von **Applikationsmetriken**, z. B. »Wie schnell wird Seite X in meiner Webanwendung im Durchschnitt geladen«, aber auch von **Prozessmetriken**.	- Applikationsmetriken - Prozessmetriken, wie z. B. bei Kanban implementiert
Sharing	Das Teilen von Werkzeugen, Metriken, aber auch von Prozessen, Erfolgen und Misserfolgen, bspw. zwischen Softwareentwicklern und Administratoren.	- Kanban[111] - Angleichung von Werkzeugen und Umgebungen in der Entwicklungsabteilung sowie Operations.

2.4.5 Ergänzende Methoden

In Tabelle 9 werden noch weitere zu untersuchende Methoden, welche die vorherigen agilen Modelle ergänzen und bei Jimdo zum Einsatz kommen, kurz vorgestellt.

Tabelle 9: Ergänzende agile Methoden

Methode	Beschreibung
SbE (Specification by Example)	Format und Methode zur kollaborativen Spezifizierung von funktionalen Akzeptanztests.[112]

[106] nach einer Kategorisierung aus Willis 2010.
[107] vgl. ebd., eine fehleroffene Kultur wird auch als Vorraussetzung für Wissensmanagement genannt, vgl. Probst u. a. 2010, S. 120
[108] vgl. Willis 2010.
[109] vgl. Humble und Farley 2010.
[110] vgl. Loukides 2012.
[111] Kanban kann die komplette Wertschöpfungskette visualisieren, welche zumeist mehrere Stationen in einem Unternehmen einschließt.
[112] Auch ATDD (Acceptance Test-driven Development) genannt, im folgenden wird SbE verwendet, vgl. dazu Adzic 2011, S. xiii.

2.5 Zusammenfassung und kritische Betrachtung

In diesem Kapitel wurden grundlegende Definitionen getroffen und in das Thema eingeführt. Die Begriffe »Wissen und Wissenstypen« wurden differenziert, dabei wurde zwischen den Wissenstypen implizit und explizit unterschieden, die Übergänge nach Nonaka und Takeuchi skizziert sowie die Idee des organisationalen Lernens eingeführt. Danach wurde der Begriff »Wissensmanagement« in Anlehnung an Willkes systemische Sichtweise als Geschäftsprozess betrachtet, bei dem aufgezeigt wird, wie personales in organisationales Wissen überführt wird. Es wurden Modelle, die für das Vorgehen in dieser Arbeit relevant sind, vorgestellt: Das eben genannte Wissensmanagement als Geschäftsprozess, das Grazer Metamodell des Wissensmanagements, Earl's schools of knowledge management sowie die Übertragung des St. Galler Managementkonzept auf Wissensmanagement durch Probst u. a. 2010.

Im Abschnitt »Software Engineering und seine Wissensbereiche« wurde Software Engineering anhand von unterschiedlichen Zugängen vorgestellt. Es wurden die »Wissensbereiche« des Software Engineerings in Anlehnung an SWEBOK benannt sowie »Arten von Wissen« (Domänenwissen, Anforderungswissen, Applikationswissen, Prozesswissen und Erfahrungswissen), auf die im späteren Verlauf eingegangen werden soll, eingegrenzt. Es folgte ein Blick auf die »Unterschiede traditioneller und agiler Softwareentwicklung«. Dieser besteht vor allem in einer unterschiedlichen Managementsicht in Hinblick auf die Unternehmenssteuerung. Es konnte eine Grobeinteilung der Managementsichten in mechanistisch / tayloristisch und systemisch / agil gezeigt werden.

Zur Orientierung wurde ein »Überblick bisheriger Forschungsansätze« zu Wissensmanagement und Software Engineering gegeben, dabei wurde der Forschungsbedarf in Bezug auf agile Methoden deutlich.

Abschließend fand eine »Kurzvorstellung agiler Softwareentwicklungsmodelle«, die bei Jimdo im Einsatz sind, statt: Scrum, Extreme Programming, Software Kanban, DevOps sowie ergänzende Methoden.

Zur kritischen Betrachtung der Vorgehensweise in diesem Kapitel ist anzumerken, dass die Auswahlentscheidung der Modelle anders hätte ausfallen können, da Wissensmanagement ein weites Feld ist und es zahlreiche Vertreter und Ansätze gibt. Die hier getroffene Auswahl an Ansätzen und Methoden für eine Einordnung und Systematisierung des Unternehmensbeispiels stammen aus dem deutsch- und englischsprachigen Raum und erlauben einen Blick auf

den internationalen Diskurs. Sie wurden vor dem Hintergrund ihrer Aktualität, ihrer Verbreitung sowie der Relevanz ausgewählt, inwiefern sie die agile Vorgehensweise, wie sie bei Jimdo praktiziert wird, mitdenken bzw. zur Einordnung dieser dienen können.

3 Agile Softwareentwicklung bei Jimdo

Dieses Kapitel beschäftigt sich mit dem Unternehmen Jimdo. Es werden das Unternehmen, die Arbeit in den Softwareteams sowie die bereits eingesetzten agilen Methoden vorgestellt. Zudem wird eine normative Einordnung des Unternehmens in Bezug auf Wissensmanagement getroffen, das im Hauptkapitel zum »Konzept eines Wissensmanagementmodells« weiterentwickelt wird.

3.1 Unternehmensvorstellung

3.1.1 Generelles zum Unternehmen

Das Unternehmen »Jimdo« gibt es in der heutigen Form seit Februar 2007 und kann damit als ein Urgestein im deutschen 'Web 2.0' Umfeld bezeichnet werden.[113]

Christian Springub, Fridtjof Detzner und Matthias Henze gründeten 2004 die Firma NorthClick, in der sie die Basis-Software für Jimdo entwickelten - ursprünglich für Firmen - gedacht, daraus entwickelte sich die Idee der kostenlosen Jimdo-Page. Hauptfirmensitz ist in Hamburg, es gibt weitere Außenstellen in Japan, China und den USA.

Heute zählen etwa 120 Mitarbeiter zum Unternehmen, von denen ca. 80 in Hamburg arbeiten.[114]

Zum Produkt: Jimdo bietet unter www.jimdo.com einen kostenlosen Homepage-Baukasten an. Ohne Vorkenntnisse und in wenigen Minuten lässt sich die eigene Online-Präsenz erstellen - inklusive Funktionen wie Online-Shop, Blog, Bildergalerien, Youtube-Videos, Mobile Ansicht u.v.m.[115]

Die Software wird als »Software-as-a-service« bereitgestellt, d. h. der Benutzer muss sich keine Software herunterladen und installieren. Das Programm läuft komplett im Webbrowser des Benutzers ab.

---

[113] vgl. URL: `http://en.wikipedia.org/wiki/Jimdo` (abgerufen am 15.11.2012)
[114] vgl. URL: `http://de.jimdo.com/presse/pressemitteilungen/presse-faq/` (abgerufen am 15.11.2012)
[115] vgl. URL: `http://de.jimdo.com/presse/pressemitteilungen/presse-faq/` (abgerufen am 15.11.2012)

Als Versionen gibt es JimdoFree, JimdoPro und JimdoBusiness, die sich in Speicherplatz sowie einigen Features und damit entsprechend preislich unterscheiden. JimdoFree ist als Grundvariante kostenlos. Inzwischen gibt es mehr als sechs Millionen erstellte Jimdo Pages, das Produkt wird in elf Sprachversionen.[116]

3.1.2 Unternehmenskultur

Die Gründer von Jimdo haben Unternehmenskultur als kritische Komponente des Unternehmenserfolgs identifiziert. So werden die Werte wie z. B. Lernen, Offenheit und Fehlertoleranz unterstrichen und sind als Kulturverfassung auch verschriftlicht. Um diese Kultur mit dem starken Mitarbeiterwachstum skalieren zu können, wurde 2011 sogar eine »Kultur-Managerin« eingestellt, welche sich z. B. um neue Mitarbeiter kümmert, oder Unternehmungen organisiert, aber auch die aktive Vermittlung der Kulturwerte vorantreibt.[117]

Es sind bereits viele Orte für informelle Treffen geschaffen worden, z. B. Großraumbüros als grundlegende Struktur sowie Kaffee- , und Sofaecken. Auch ein eigenes Restaurant, in dem jeden Tag Mittagessen serviert wird, wurde 2012 in Betrieb genommen. Einmal pro Woche gibt es ein gemeinsames Mitarbeiterfrühstück. Nach der »spatial school« von Earl 2001 stellt dies schon gelebte Wissensmanagementaktivitäten dar.

Einen Eindruck der Unternehmenskultur bekommt man im Video »Unternehmenskultur bei Jimdo - Feel Good, Wenig Schlecht!«[118], welches Jimdo 2012 produziert hat, um die gelebte Unternehmenskultur einer breiten Öffentlichkeit vorzustellen.

[116] vgl. URL: http://de.jimdo.com/presse/pressemitteilungen/factsheet/ (abgerufen am 15.11.2012)
[117] vgl. URL: http://www.startupcareer.de/9411/unternehmenskultur-sprintmanager-personalentwicklung-jimdo/ (abgerufen am 15.11.2012)
[118] URL: http://vimeo.com/53967812 (abgerufen am 15.11.2012)

3.2 Software Teams

3.2.1 Struktur und Organigramm

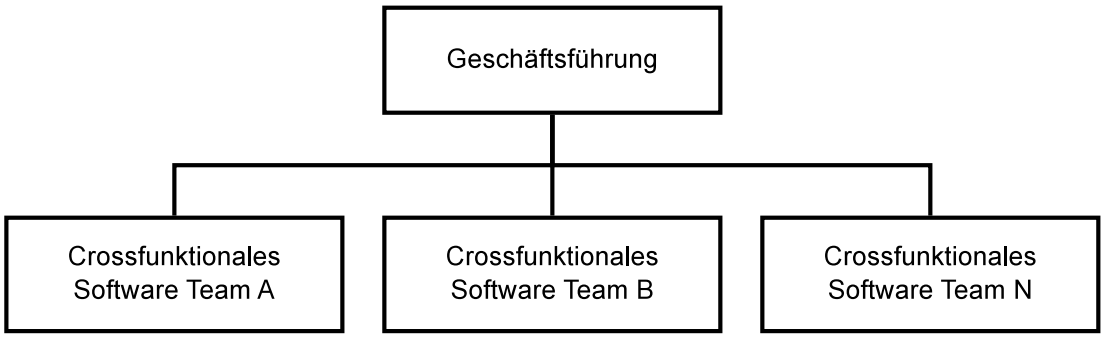

Abbildung 5: Organigramm der Software Teams bei Jimdo

Abbildung 5 zeigt als Organigramm (bzw. den relevanten Ausschnitt) die Struktur der Software Teams bei Jimdo. Alle Software Teams sind in Hamburg ansässig und direkt der Geschäftsführung unterstellt. Sie organisieren sich selbst, d. h. es gibt keine durch die Geschäftsführung zugewiesen Rollen oder Positionen.

Die Software Teams sind bereits teilweise crossfunktional, d. h. mit den nötigen Kompetenzen ausgestattet, um die ihnen gestellte Aufgabe erfüllen zu können. Bei Jimdo sind die Teams größtenteils so organisiert, dass sie jeweils für Teilprodukte der Jimdo Software zuständig sind. So gibt es z. B. ein Shop-Team, welches sich um die Weiterentwicklung der Shopkomponente kümmert.

Zum Wissensaustausch zwischen Teams gibt es teilweise selbstgegründete CoP (Community of Practice), die sich regelmäßig treffen.

Im Unternehmen Jimdo sind schon zahlreiche elektronische Kommunikations- und Kollaborationstools, die der freiwilligen Nutzung unterliegen, im Einsatz:

- E-Mail und Verteiler
- Ticket-System für Feature- und Bug-Tracking
- Internes Microblogging, z. B. zur Verbreitung von News[119]
- Versionskontrolle von Software
- Wiki zur Dokumentation
- Jabber-Chat mit teamübergreifenden oder teaminternen Chaträumen

[119] URL: `http://www.yammer.com/` (abgerufen am 15.11.2012)

Das heißt, viele grundlegende Werkzeuge, die auch zum Wissensmanagement beitragen[120], werden bereits benutzt. Sie sollen hier nicht weiter beleuchtet werden, da der Schwerpunkt der Arbeit auf der Untersuchung agiler Modelle liegt.

3.2.2 Eingesetzte agile Methoden

Die Teams bei Jimdo setzen unterschiedliche Methoden der vorgestellten agilen Modelle ein. Welche dies sind, ist je Team unterschiedlich und nicht vorgeschrieben. Als allgemeine Schnittmenge ergeben sich jeweils Kanban als Prozesswerkzeug sowie Daily Meeting und regelmäßige Retrospektiven. Alle anderen Methoden werden unterschiedlich (stark, häufig und regelmäßig) eingesetzt.

Tabelle 10 zeigt die Schnittmenge aller bei Jimdo derzeit eingsetzten agilen Methoden in der Übersicht. Diese Methoden dienen auch als Untersuchungsgrundlage für das nächste Kapitel.

Tabelle 10: Eingesetzte agile Methoden bei Jimdo

Modell	Bei Jimdo eingesetzte Methoden
Scrum	Effort Estimation, Sprint Planning Meeting, Daily Meeting, Sprint Retrospective (Retrospektive), crossfunktionale Teams.
XP	TDD, Refactoring, Pair Programming, Collective code ownership, CI, On-site Customer, Coding Standards.
Kanban	Visualize Workflow, Limit Work-in-Progress, Measure and Manage Flow, Explicit Process Policies
DevOps	Continuous Delivery, Infrastructure as Code, Applikationsmetriken
Ergänzende Methoden	SbE

3.3 Zusammenfassung und normative Einordnung

Nach einer Unternehmensvorstellung und einem Überblick über die Unternehmensdaten wurde die Struktur der Software Teams vorgestellt. Es wurde erwähnt, dass Teams bei Jimdo bereits selbstorganisiert bzw. -gesteuert arbeiten. Die sich schon im Einsatz befindlichen agilen Methoden wurden kurz erwähnt und somit das Untersuchungsfeld für das nächste Kapitel ausgewiesen.

Im nächsten Kapitel soll für die Software Teams bei Jimdo ein Wissensmanagementkonzept entworfen werden. Vorbereitend dazu muss das Unternehmen

[120] vgl. North 2011, S. 316.

normativ eingeordnet werden, um eine möglichst passende Vorgehensweise entwickeln zu können. Die normative Ebene des Wissensmanagements bildet sich laut Probst u. a. aus »wissensorientierter Perspektive relevanten unternehmenspolitischen und -kulturellen 'Leitplanken' des Managements«[121]. Jimdo versteht sich als agiles Unternehmen. In Kapitel 2 wurde festgestellt, dass sich hierfür ein personenorientiertes Wissensmanagement anbietet. Da Jimdo bereits sehr auf selbstgesteuerte bzw. organisierte Teams setzt, wird ein systemischer Wissensmanagementsansatz zugrunde gelegt, welcher ebenfalls auf Selbst- bzw. Kontextsteuerung angelegt ist.

Abbildung 6 zeigt diese Einordnung in der Übersicht. Auf normativer Ebene wurde aus dem Unternehmenskontext heraus ein personenorientiertes, systemisches Wissensmanagement festgestellt bzw. festgelegt und somit die Leitplanken für die Ausgestaltung der strategischen und operativen Ebene gesetzt. Kapitel 4 befasst sich der Konzeption der beiden noch fehlenden Ebenen.

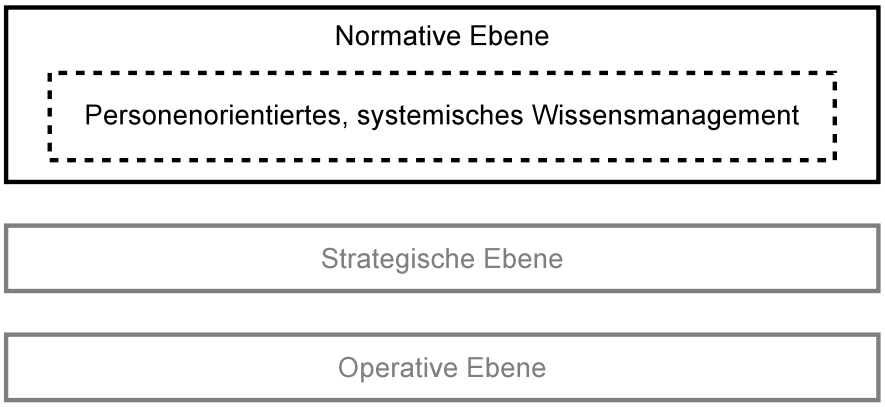

Abbildung 6: Normative Ausrichtung des Wissensmanagements, Schritt 1

[121] Probst u. a. 2010, S. 41.

4 Konzept: Entwicklung eines Wissensmanagementmodells

Das Hauptkapitel dieser Arbeit beginnt mit einer Einordnung der agilen Methoden, inwiefern sie schon implizit Wissensmanagement »betreiben«, ohne als explizites Wissensmanagementinstrument installiert worden zu sein. Auf Grundlage der normativen Einordnung im vorherigen Kapitel, wird das - auf agilen Methoden basierende - »Wissensmanagement als Geschäftsprozess« auf strategischer und operativer Ebene ergänzt und angepasst. Grundgedanke ist dabei, ein für Jimdo optimales Wissensmanagementmodell zu entwerfen.

4.1 Einordnung agiler Methoden

Die Leithypothese dieser Studie besagt, dass agile Methoden implizit ein ausreichendes Wissensmanagement mit sich bringen. Um dies zu untersuchen, werden in diesem Abschnitt die ausgewählten agilen Methoden systematisiert, um herauszufinden, ob oder inwiefern sie Wissensmanagementaktivitäten unterstützen.

Als Basis der folgenden Überlegungen dient der systemische Wissensmanagementansatz von Willke. Er versteht Wissensmanagement als Geschäftsprozess, innerhalb dessen Wissen entwickelt, expliziert, kodifiziert, verteilt, genutzt und revidiert wird, wie in Abbildung 2 dargestellt wurde. Willke selbst führt das Beispiel des Instruments »Mikroartikel« an und zeigt, wie dabei die Stufen des Wissensmanagements als Geschäftsprozess durchlaufen werden, so dass eine zunächst individuelle Lernerfahrung (personales Wissen) zum Lernen des sozialen Systems (organisationales Wissen) beiträgt.[122]

Im Folgenden werden Methoden der agilen Softwareentwicklung daraufhin untersucht, inwiefern sie die Stufen des Wissensmanagements als Geschäftsprozess umfassen. Anders als bei Willke, wird zwischen den Arten des Wissens, die für das Software Engineering als besonders kritisch betrachtet werden, unterschieden (Domänenwissen, Anforderungswissen, Applikationswissen, Prozesswissen, Applikationswissen und Erfahrungswissen), um eine differenzierte

[122] vgl. Willke 2001, S. 107 f.

Betrachtung zu ermöglichen. Dies spielt auch bei der späteren Modellentwicklung eine Rolle.

Bei der nachstehenden Betrachtung von agilen Methoden, die durch ihren kommunikativen Ansatz, gemeinsames Handeln und das Teilen von Erfahrungen geprägt sind, wird davon ausgegangen, dass, im Sinne der »Socialization« von Nonaka und Takeuchi implizites Wissen ohne eine Verschriftlichung oder anderweitige Dokumentation (bei Willke »Generalisieren« = explizierte und dokumentierte Form des Wissens[123]) geteilt werden kann. Verbale Kommunikation und einfaches »Zeigen« jeder Art (beobachten, imitieren und handeln) trägt zur Verteilung von Wissen bei[124].

4.1.1 Generierung

Der erste Schritt im Modell bei Willke[125] macht die »Generierung von Wissen« aus, welches in den meisten Fällen schon vorhanden ist. Im Beispiel des Mikroartikels wird Wissen auf der Basis einer Lernerfahrung entwickelt: Am Anfang steht eine Idee, eine Konzeption, eine Einsicht u. ä.[126] Es geht also darum, zu untersuchen, wann mithilfe agiler Methoden eine Lernerfahrung gemacht werden kann.

Die Diversität der Fähigkeiten der Mitglieder in crossfunktionalen Teams bietet positive Bedingungen für die kreative Zusammenarbeit und trägt zur Entwicklung von neuem Wissen bei.[127] Benötigtes Wissen ist stets im Team vorhanden und ein Wissenstransfer kann bei Bedarf aktiviert werden bzw. ergibt sich im Prozess.[128]

Im Rahmen von Software Kanban fördert die Limitierung von WIP die Transparenz und die Kommunikation über den Prozess, z. B. werden Probleme wie Engpässe oder auch Leerläufe von »Stationen« schnell sichtbar. Die Mitarbeiter müssen ihr Prozesswissen gemäß der Prioritäten und erwarteten Effekte explizieren, mit dem Team teilen und es ggf. gemeinsam entwickeln.

[123] vgl. Willke 2001, S. 87.
[124] vgl. Nonaka und Takeuchi 1995, S. 62 f.
[125] Willke 2001, S. 87.
[126] vgl. ebd., S. 107.
[127] vgl. Nonaka und Takeuchi 1995, S. 76 ff.; Probst u. a. 2010, S. 128.
[128] vgl. Chau u. a. 2003, S. 306; vgl. Nonaka und Takeuchi 1986, S. 7.

Durch die Explizierung von Wissen über Effizienz und Effektivität des Prozesses bei der Erstellung von Prozessmetriken, z. B. »Lead Times« oder CFD, kann ebenfalls neues Prozesswissen entwickelt werden.[129]

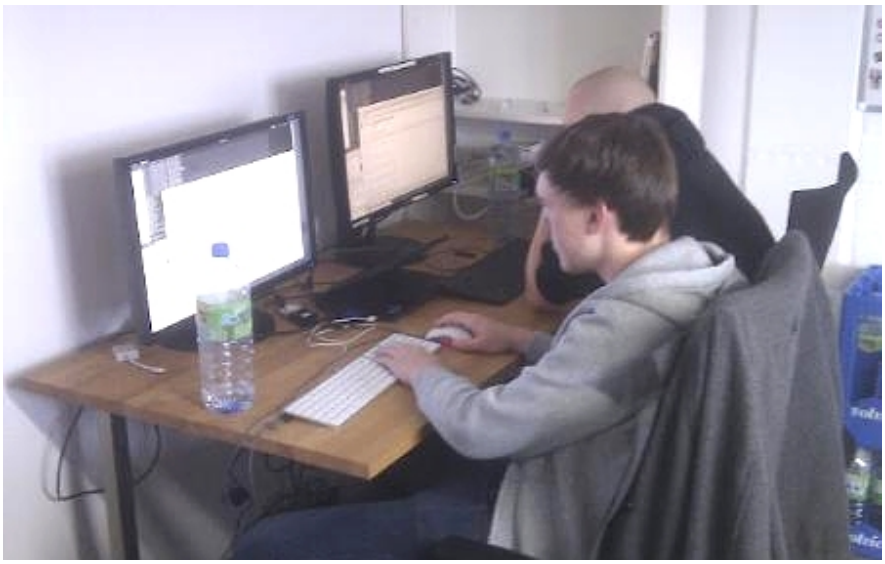

Abbildung 7: Pair Programming Arbeitsplatz: Jeweils zwei (geklonte) Monitore, Tastaturen und Mäuse

Abbildung 7 zeigt ein typisches Setup zum Pair Programming: Zwei Softwareentwickler sitzen an einem Computer, haben aber jeweils einen eigenen Bildschirm (mit geklontem Inhalt), eine eigene Maus und Tastatur. Dies erlaubt ein komfortables kollaboratives Arbeiten an einer Aufgabe. Beim Pair Programming befinden sich beide Paarteile ständig im Brainstorming[130] und somit in einem ständigen Abstimmungsprozess, der neues Wissen jeglicher Art hervorbringen kann.[131]

Auch die anderen kommunikativen Methoden wie z. B. Daily Meeting, Planning Meeting, Retrospektiven können Auslöser für die Generierung von Wissen im Sinne einer Idee oder Lernerfahrung sein. Jegliche Form der Verbalisierung von Gedanken regt die Neuordnung von Informationen an, so kann allein durch das Aussprechen schon eine neue Idee entwickelt werden. Eine besondere Form ist beispielsweise das »rubberduck debugging«, d. h. wenn man keinen menschlichen Kommunikationspartner hat, so erzählt man seinen

[129] vgl. Anderson 2010, S. 144.
[130] vgl. Williams und Kessler 2003, S. 26.
[131] Brainstorming ist allgemein als Kreativtechnik angesehen, dabei wird angestrebt »durch Sammeln von spontanen Einfällen [...] die [beste] Lösung für ein Problem zu finden« (vgl. Probst u. a. 2010, S. 120).

Gedanken oder sein Problem einfach einer Gummiente. Alleine das Aussprechen des Problems kann helfen, selbst auf die Lösung zu kommen.[132]

4.1.2 Topographien / Explikation

Willke nennt diesen Schritt in seinem Modell »Topographie des Wissens«, dabei geht es ihm um eine »Bestandsaufnahme des vorhandenen und des fehlenden Wissens.«[133] Dazu gehört die Einordnung des Wissens in einen sinnvollen Kontext, denn neue Informationen führen nicht per se zu neuem Wissen, sondern nur, wenn sie für das System einen Sinn ergeben bzw. in einen sinnvollen Kontext eingeordnet werden können.[134] Wissen muss also von den Personen in der Weise expliziert werden, dass die Kontextanknüpfung gegeben ist. Die individuellen Vorstellungsbilder im Kopf müssen abgeglichen werden. Diesen Effekt bezeichnen auch Nonaka und Takeuchi als Redundanz.[135] Über die Wissenslandkarten in den Köpfen muss Konsens bestehen, welche im Unternehmen vom Unternehmenskontext und den Unternehmenszielen geprägt werden können.

Übertragen auf den Softwareentwicklungsprozess geht es an dieser Stelle um Aktivitäten, die Wissen explizieren sowie die eine gemeinsame Sinnbildung betreffen. In Bezug auf die Erkennung von Nichtwissen bzw. Wissenslücken setzen crossfunktionale Teams an. Die Teammitglieder sind Experten in ihrem Bereich, haben aber auch ein breites Oberflächenwissen zu anderen Gebieten (sog. »T-shape skills«[136]). Die Frage ist, inwiefern davon ausgegangen werden kann, dass Mitglieder dieser Teams sich selbstständig weiterbilden und ihre blinden Flecken erkennen[137] oder das Management durch Personalentwicklung die Mitarbeiter unterstützten sollte. Dies wird an späterer Stelle in dieser Arbeit noch diskutiert.

Beim Pair Programming unterstützt Pair Negotiation den Prozess der Explikation: Paare müssen sich beim Programmieren auf einen Weg verständigen und gleichen somit implizit ihre Wissenslandkarten ab. Darüber lernt das Individu-

[132] vgl. Bryant u. a. 2006, S. 54 f.
[133] Willke 2007, S. 109.
[134] vgl. ebd., S. 45.
[135] vgl. Nonaka und Takeuchi 1995, S. 81.
[136] vgl. Rubin 2012, S. 201-202.
[137] sog. »Positive Ignoranz« nach Schneider 2006

um, sich selbst besser einzuschätzen.[138] Gegebenenfalls werden ihm sogar Wissenslücken klar, die es selbstständig schließen kann.

Die kommunikativen Anteile agiler Methoden unterstützen den Abgleich von individuellen Vorstellungsbildern im Kopf und sorgen auf diese Weise für weniger Missverständnisse, Wissenslandkarten in den Köpfen gleichen sich an.

Die Explikation von Anforderungswissen kann über User Stories und Szenarien (SbE) passieren. User stories und die Anwesenheit des Kunden fördern durch verbale Kommunikation die Entwicklung von Anforderungswissen. So liegen User stories oft in dem Format »Als <Rolle> möchte ich <Feature>, so dass <Wertschöpfung für das Unternehmen>« vor[139], welches die Explikation der Intention des Kunden fördern kann. Die Explikation und Verteilung von Anforderungswissen mittels SbE wird im nächsten Abschnitt vorgestellt.

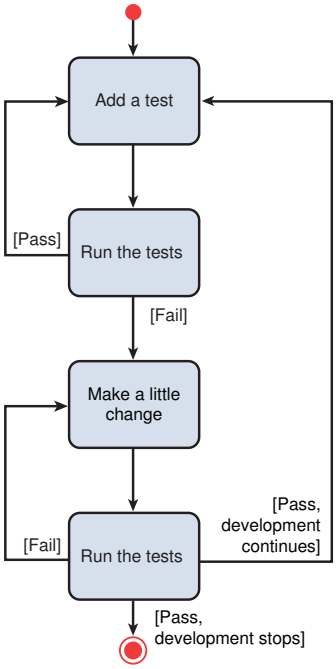

Abbildung 8: Schritte von TFD (Ambler und Lines 2012, S. 349)

Abbildung 8 zeigt die Explikation von Wissen (Applikationswissen, Anforderungswissen und Domänenwissen) in ausführbare Spezifikationen mittels Unit tests: Bevor ein Softwarentwickler einen Teil des Programms implementiert, schreibt er einen Test und überprüft, ob dieser fehlschlägt. Dann implementiert er die neue Funktionalität und überprüft, bis der Test nicht mehr fehlschlägt

[138] vgl. Williams und Kessler 2003, S. 25.
[139] vgl. Cohn 2006, S. 25 f.

(Test pass). Dann beginnt der Kreislauf von neuem: Entweder wird ein neuer Test geschrieben, oder der Quelltext wird durch Refactoring optimiert.

Der Vorteil dieser Methode ist, dass jeder Gedanke erst expliziert werden muss, bevor er implementiert wird, erst dadurch wird das Wissen greifbar und sortierbar.[140] Weiterhin wird keine Funktionalität implementiert, die gerade nicht gebraucht wird, und es wird sichergestellt, dass das gesamte System durch Tests spezifiziert wurde.[141] Pair Programming verstärkt diesen Effekt, da das explizierte Wissen direkt mit dem Partner abgeglichen und somit revidiert wird.

4.1.3 Kodifizierung und Aufbereitung

Willke nennt diese Stufe in seinem Modell »Dokumentation von Wissen«. Dabei wird relevantes Wissen über eine Dokumentationsform, in seinem Beispiel der Mikroartikel, verfasst und veröffentlicht, so dass die Wissensinhalte eingespeist, von anderen genutzt und kommentiert werden können.[142] Im Kontext agiler Methoden wird in diesem Schritt der Wissensbewahrung von »Kodifizierung und Aufbereitung« ausgegangen, da das Wissen nicht lediglich dokumentiert, sondern vielmehr, falls möglich, in maschinenlesbare, ausführbare Spezifikationen gebracht werden soll[143].

Wissensbewahrung bei agilen Methoden findet zum einen auf persönlicher und zum anderen auf technischer Ebene statt. Durch die Kommunikation zwischen Personen wird das Wissen »in den Köpfen« aktiv und aktuell gehalten, z. B. beim Pair Programming und in Daily Meetings.

Pair Programming und die damit zusammenhängende Redundanz von Wissen trägt dazu bei, das Risiko durch Ausfall von Mitarbeitern zu minimieren, denn implizites Wissen ist immer in mindestens zwei Köpfen vorhanden. Durch Pair Rotation kann die Redundanz sogar noch erhöht werden.[144]

Auf der technischen Ebene wird durch Automatisierung Anforderungswissen und Applikationswissen gespeichert. TDD und SbE sind ausführbare Spezifi-

[140] vgl. Willke 2007, S. 73.
[141] vgl. Beck und Andres 2005, S. 31.
[142] vgl. Willke 2007, S. 110.
[143] »Secondly, in ATDD each and every functional requirement is an up-to-date executable test.« (Haugset und Stalhane 2012, S. 5296)
[144] vgl. Williams und Kessler 2003, S. 77.

kationen[145] und bewahren Wissen durch die Validitätsprüfung (zumeist mithilfe eines CI-Systems[146]). Mittels SbE kann eine sog. »living documentation« inkl. einer »Ubiquitous Language«, erzielt werden, d. h. eine einheitliche Dokumentation, die gleichzeitig Akzeptanztests und ein Glossar beinhaltet.[147] Erstellung, Nutzung und Pflege von Spezifikationen, Glossar und Akzeptanztests sind somit in den Prozess integriert und die Aufgabe aller Teilhaber (Kunde, Team).

Das Konzept der »agilen Dokumentation« sieht vor, dass ausführbare Spezifikationen immer vor statischen Dokumenten bevorzugt werden sollen.[148] Ein Vorteil besteht im »Single Sourcing«: Wenn Anforderungswissen durch Tests (z. B. Akzeptanztests oder Unit tests) spezifiziert wurde, zeigen diese Spezifikationen den derzeitigen »Single point of truth«, also den aktuellen Stand.[149] Statische Dokumente sind nicht in dieser Art und Geschwindigkeit aktuell zu halten und müssen zudem in einem weiteren Arbeitsschritt erstellt und gepflegt werden.

CI-Systeme speichern Metriken wie Lines Of Code, Code Coverage, zyklomatische Komplexität und erlauben dadurch Langzeitanalysen.[150] Applikationswissen wird also über einen längeren Zeitraum bewahrt.

Über den Einsatz von Kanban wird Prozesswissen dokumentiert und bewahrt. Dabei wird über Explicit Process Policies definiert, wann ein ToDo abgeschlossen ist, oder welche Schritte befolgt werden müssen, um eine Aufgabe zu komplettieren. Durch Explizierung kann auch eine Überprüfung der Sinnhaftigkeit von Prozessschritten aktiviert werden.[151] Dies ist auch eine wichtige Systemeigenschaft im Sinne von Königswieser und Hillebrand 2005, da das Team eine Objektivierung von Vereinbahrungen vornimmt.

Im Bereich Configuration management bzw. Operations setzen Konzepte wie »Continuous Delivery« und »Infrastructure as Code« an. Wissen über Konfiguration (Applikationswissen) der Releases/ Deployment von Software (Prozess-

[145] vgl. Adzic 2011, S. 21; Ambler und Lines 2012, S. 350.
[146] CI-Systeme sind spezielle Software, die regelmäßig automatisiert Unit tests und automatisierte Akzeptanztests ausführt. Das Ergebnis wird dem Entwicklungsteam z. B. per E-Mail oder Chat zurückgemeldet, vgl. Paul 2007.
[147] vgl. Adzic 2011, S. 7-8,188.
[148] vgl. Ambler 2012b.
[149] vgl. hierzu Ambler 2012a: »Developers rarely trust the documentation, particularly detailed documentation because it's usually out of sync with the code.«
[150] »Continuous Inspection«,vgl. Paul 2007
[151] vgl. Anderson 2010, S. 38-39; Willke 2007, S. 73.

wissen) wird z. B. in Programmen (»Scripts«) kodifiziert und ist somit wiederum eine ausführbare Spezifikation, die nicht veralten kann. Weiterhin werden Fehler, die durch Nichtbefolgung von Schritten einer Dokumentationen entstehen, vermieden.[152]

4.1.4 Verteilung

Auf dieser Stufe spricht Willke von der »Verteilung von Wissen«. Er plädiert dafür, dass ein möglichst offener und uneingeschränkter Zugang zu wichtigen Informationen besteht, gerade in Unternehmen, für die Wissen eine wichtige Ressource ist.[153] Denn Wissen steht in Verbindung mit einem spezifischen praktischen Kontext und Erfahrungen, die durch Informationsweitergabe trotzdem bestehen bleiben. Zu seinem Beispiel des Mikroartikels nennt er auf dieser Stufe das Publizieren des Textes, so dass er für andere zugänglich wird[154].

Aufgrund ihrer kommunikationsfördernden Eigenschaften sind agile Methoden in der Verteilung von Wissen innerhalb von Teams besonders wirksam, da direkte Kommunikation besonders verlustfrei ist. Gerade komplexe Zusammenhänge können eher mündlich eher korrekt übertragen werden als durch Dokumente.[155]

Crossfunktionale Teams tragen zur Wissensverteilung bei, denn benötigtes Wissen ist stets im Team vorhanden und bei Bedarf kann der Wissenstransfer aktiviert werden[156].

Durch Standup Meetings bzw. Daily Meetings wird Wissen über den Fortschritt bzw. Zustand von Tasks sowie aufgekommene Probleme expliziert und verteilt.[157] Diese Meetings finden zumeist vor einem Task-Board statt, welches in diesem Zuge aktualisiert wird. Auf diesem Wege wird ebenfalls Prozess- und Projektwissen verteilt.

[152] vgl. Humble und Farley 2010, S. 6.
[153] vgl. Willke 2007, S. 111.
[154] vgl. Willke 2001, S. 108.
[155] vgl. Melnik und Maurer 2004.
[156] vgl. Chau u. a. 2003, S. 306; Nonaka und Takeuchi 1986, S. 7.
[157] vgl. Melnik und Maurer 2004, S. 23.

Spontane Treffen nach dem Daily Meeting sind erwünscht, hier können zusätzliche oder offene Details aus dem Daily Meeting geklärt werden, es kann also Wissen jeglicher Art verteilt werden.[158]

Auf Kanban Boards sind Auslastung, Zustand und Veränderungen für Team-Mitglieder und Außenstehende ständig sichtbar und somit expliziert.[159] Unterstützend dabei sind visuelle Merkmale wie Kartengröße für Komplexität der Aufgabe, und verschiedene Farben für Art der Aufgabe[160].

Domänen- und Anforderungswissen kann z. B. durch Techniken wie Planungspoker (Effort estimation, Planning meeting) kollaborativ expliziert und verteilt werden: Mit dem gesamten Team wird der Aufwand für Anforderungen geschätzt: Hier wird aktiv Wissen ausgetauscht, da die Teammitglieder begründen müssen, dass etwas z. B. einen sehr großen Aufwand mit sich bringt; dazu müssen sie ihr Wissen darüber explizieren. Auch spezielles Wissen aus Design, Construction oder Testing kann somit aktiviert werden.[161]

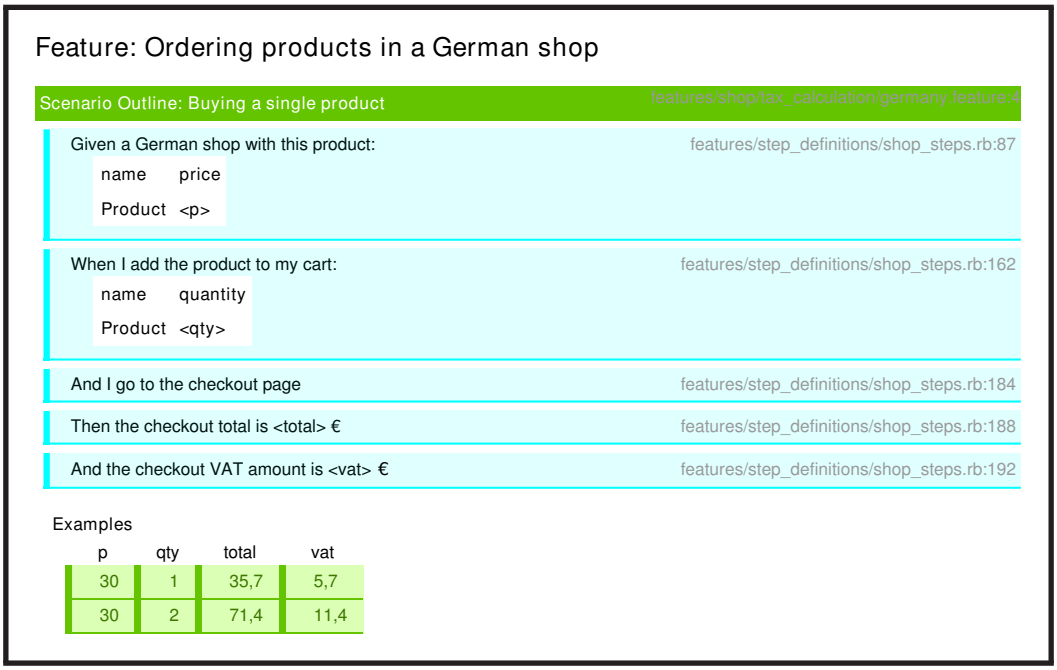

Abbildung 9: Beispielszenario einer Softwareanforderung im Jimdo-Shop

[158] vgl. Anderson 2010, S. 82 f.
[159] vgl. ebd., S. 65.
[160] vgl. ebd., S. 125.
[161] vgl. Cohn 2006, S. 59.

SbE fördert Verteilung von Wissen von Softwareanforderungen: Einerseits erfolgt die Analyse kooperativ, z. B. durch »Pair-writing«[162], andererseits zwingt die Angabe von Beispielen den Kunden, seine Anforderungen zu abstrahieren und damit auch zu reflektieren. Durch die direkte Kommunikation und kollaborative Spezifizierung können Missverständnisse ausgeschlossen werden[163]. Abbildung 9 zeigt ein Beispiel für eine Anforderung des Jimdo Shop Moduls: Es wird abstrakt die Berechnung der Mehrwertsteuer (VAT) spezifiziert, ohne Implementierungsdetails zu verlangen. Das Format ist gleichzeitig gut für Menschen lesbar (Given/When/Then), andererseits auch durch Maschinen lesbar und damit als DSL für automatisierte sowie ausführbare Spezifikationen geeignet[164], was später noch gezeigt werden soll.

Pair Programming unterstützt die Wissensverteilung gleich in Bezug auf mehrere Faktoren: Wissen wird andauernd zwischen den Paarteilen ausgetauscht, dies betrifft nicht nur Erfahrungswissen, wie bspw. eine Tastenkombination im Editor, sondern auch Applikationswissen wie Coding Standards. Pair Programming ist eine Art »training-on-the-job«: Neue Teammitglieder werden schnell eingearbeitet und sind sofort produktiv. Verstärkt wird die Verteilung von Wissen im Team durch »Pair rotation«, d. h. die Paare wechseln regelmäßig, so dass Wissen letztendlich im gesamten Team verteilt wird.[165]

4.1.5 Nutzung

Bei der Stufe der »Nutzung von Wissen« weist Willke auf die Problematik von fehlender Akzeptanz hin. Die bloße Bereitstellung von Wissen genügt nicht, um eine Nutzung zu initiieren. Die Vorteile der Nutzung von Wissen und der Dokumentationsformen etc. müssen für die Mitarbeiter vorhanden sein (bspw. Zeitersparnis, relevante und aktuelle Daten) und nachvollziehbar sein. Chau u. a. betonen, dass die Nutzung von Wissen stark mit Vertrauen und gegenseitigem Respekt zusammenhängt.[166] Fehlt das Vertrauen, z. B. in die Aussage eines Teamkollegen oder in die Programmierung, stellt dies eine Wissensbarriere für

[162] vgl. Adzic 2011, S. 77.
[163] vgl. Haugset und Stalhane 2012, S. 5294.
[164] Die im Beispiel benutzte DSL (Domain specific language) ist »Gherkin«, vgl. URL: `https://github.com/cucumber/cucumber/wiki/Gherkin` (abgerufen am 15.11.2012)
[165] vgl. Williams und Kessler 2003, S. 29; vgl. Chau u. a. 2003, S. 305.
[166] vgl. ebd., S. 305.

die Nutzung des vorhandenen Wissens dar.[167] Den Erfolg des Mikroartikels macht Willke davon abhängig, wer und wie viele andere Personen ihn lesen und nutzen. Indikator für die Nutzung ist bei ihm die Zitation.[168]

Agile Methoden bauen auf mehrere Weisen Vertrauen auf und erleichtern somit die Wissensnutzung. Die ständige direkte mündliche Kommunikation durch Daily Meetings, Retrospektiven, Pair Programming und On-site Customer trägt zur Vertrauensbildung bei.[169] Beim Pair Programming kann neu gelerntes Wissen direkt genutzt werden, regelmäßiges Wechseln von Driver und Navigator führen zu einer schnellen Internalisierung von neu gelerntem Wissen. Durch gegenseitiges Vertrauen (»pair trust«) wird die Teambindung stärker, welches eine noch weitergehende Wissensnutzung ermöglicht.[170]

Collective code ownership und ausführbare Spezifikationen und TDD unterstützen das Vertrauen der Softwareentwickler in die Software. Wenn die Software durchgängig durch Tests spezifiziert ist, steigert dies das Vertrauen in bestehende Funktionen und Komponenten. Dies führt zu einer erhöhten Wahrscheinlichkeit der Wiederverwendung von Komponenten und Verringerung des NIH-Syndroms.[171]

Gespeichertes Anforderungswissen wird, wie beschrieben, mittels SbE zu einer lebendigen Dokumentation, die auch von anderen Stellen, Abteilungen oder Teams im Unternehmen benutzt werden kann.[172]

Als eine spezielle Form der Wissensnutzung kann man die Automatisierung von Unit- und Akzeptanztests sehen: Ändert sich eine Anforderung und damit die Tests, ohne dass die Implementierung verändert wird, so meldet das CI-System dem Team oder einem einzelnen Entwickler, dass ein oder mehrere Tests fehlgeschlagen sind. Das gleiche gilt für versehentliche Änderungen (Regression testing[173]), durch die eine Anforderung nicht mehr erfüllt ist: Einmal gespeichertes Anforderungswissen wird regelmäßig validiert, und »meldet« sich, falls die Validierung fehlschlägt, und wird somit zurück in das Gedächtnis gerufen. Automatisierte Tests bieten ein zusätzliches Sicherheitsnetz

[167] vgl. Chau u. a. 2003, S. 305; Schneider 2009, S. 160.
[168] vgl. Willke 2001, S. 108.
[169] vgl. Chau u. a. 2003, S. 305.
[170] vgl. Williams und Kessler 2003, S. 29 f.
[171] vgl. Beck und Andres 2005, S. 51.
[172] vgl. Adzic 2011, S. 191 ff.
[173] vgl. Abran u. a. 2004, 5–4 f.

zu manuellen Tests, es sollte allerdings kritisch hinterfragt werden, inwiefern man sich komplett auf automatisierte Tests verlassen kann oder sollte.[174]

4.1.6 Revision

Ein Wissensmanagementprozess ist erst abgeschlossen bzw. durchläuft den Kreislauf erneut, wenn das Wissen und die Ziele des Wissens vor dem Hintergrund der strategischen Ziele des Systems abgeglichen wurden. Die Stufe der »Revision des Wissens«, wie sie bei Willke bezeichnet wird, umfasst »die kontinuierliche und organisierte Revision des vorhandenen Wissens« und auch »organisiertes Entlernen, Verlernen [...] und Vergessen«[175]. Die Revision wird von der Fragestellung geleitet, wozu welches Wissen gebraucht wird und inwiefern Mehrwert für die Organisation durch welches neue und revidierte Wissen erzeugt wird.[176] Im Beispiel des Mikroartikels geht es in diesem Schritt der Revision um die Öffnung für Kritik und Nachfragen und die Möglichkeit der Erweiterung des Wissens.[177]

Bei der Revision von Wissen sind agile Methoden aufgrund ihrer kommunikationsfördernden Eigenschaften innerhalb von Teams besonders wirksam. In Daily Meetings und Retrospektiven werden regelmäßig Wissensstände abgeglichen und somit revidiert. Verbesserungspotential in Bezug auf den Prozess (Prozesswissen) kann aufgedeckt werden, z. B. dass zu lange Standup Meetings abgehalten werden.[178] Über das Pair Programming findet durch »Pair negotiation« ein Abgleich von Ansichten, z. B. Problemlösungsstrategien, statt.

Durch Applikationsmetriken lässt sich neues Anforderungswissen, z. B. über Nutzungsverhalten von neu entwickelten Features, oder aber auch über die aktuelle Gesundheit der Applikation entwickeln. Diese Informationen sollten für alle Teammitglieder über einem großen Monitor (»Dashboard«) offen und gut sichtbar sein. Über die Applikationsmetriken sind ebenfalls Langzeitentwicklungen zu erkennen, z. B. ob eine Funktion der Software über die Zeit immer weniger genutzt wird. Es kann sogar geprüft werden, ob eine gewisse Stel-

[174] vgl. Haugset und Stalhane 2012, S. 5296.
[175] Willke 2007, S. 111.
[176] vgl. Willke 2001, S. 87.
[177] vgl. ebd., S. 108.
[178] vgl. Chau u. a. 2003.

Abbildung 10: Live-Applikationsmetriken der Jimdo Software

le im Quelltext überhaupt noch in der Produktionsumgebung ausgeführt wird. Abbildung 10 zeigt ein solches Dashboard.

Teams können sich selbst ein Dashboard zusammenstellen und somit immer auf dem aktuellen Wissenstand zu der von ihnen entwickelten Software bleiben. An dieser Stelle fließt sozusagen Anforderungswissen über Abteilungen hinweg: Die entwickelte Software wird für das Entwicklungsteam lebendig, Probleme, die durch Softwareänderungen ausgelöst werden, können schneller

identifiziert und falsch analysierte Anforderungen können frühzeitig erkannt werden.[179]

Durch konsequentes Refactoring[180] bleibt Software veränderbar und wartungsfähig. Wird dies vernachlässigt, so besteht die Gefahr, dass immer mehr »Technical Debt«[181] aufgebaut wird und damit »spezielles Wissen« bewahrt werden muss (»software aging«[182]). In der agilen Softwareentwicklung wird durch Refactoring angestrebt, Komplexität und Risiko in Wartung bzw. Weiterentwicklung zu vermeiden. In der konsequenten Refaktorisierung von Software besteht also eine Art des Managements von »Nichtwissen-müssen«. Mit Bezug auf Willkes Leitfrage, des organisationalen Mehrwertes von Wissen, wird geprüft, ob das spezielle Wissen dazu dient, Kernkompetenzen zu verbessern, oder aber »überflüssiges« Wissen ist, dass im Sinne der Komplexitätsreduzierung entbehrlich ist.[183] Man könnte Refactoring oder den Abbau von Technical Debt also auch als »organisiertes Entlernen« bezeichnen.[184]

Technische Exzellenz, die laut Agilem Manifest angestrebt werden sollte, stellt eine günstige Voraussetzung für Refactoring dar: Wenn ein (unerfahrener) Softwareentwickler nicht weiß, dass er gerade »Technical Debt« (»Reckless/ Inadvertent Technical Debt«[185]) erzeugt, so fehlt auch die Einsicht, dass er die »Schulden« durch Refactoring auch wieder abbauen muss[186].

4.1.7 Beispiele

In der Abbildung 11 sind drei Beispiele zusammengefasst, wie Wissensmanagement als operativer Geschäftsprozess durch agile Methoden unterstützt wird. Der Geschäftsprozess soll, trotz der tabellarischen Darstellungsweise, als Kreislauf gedacht werden. In der linken Spalte sind die Stufen des Wissensmanagements als Geschäftsprozess nach Willke aufgeführt.

---

[179] vgl. Ambler und Lines 2012, S. 236; Allspaw und Robbins 2010, S. 236 f.; Anbieter von Werkzeugen für Softwaremetriken in Webanwendungen sind z. B. Librato (URL: `https://metrics.librato.com/` (abgerufen am 15.11.2012)) und NewRelic (URL: `http://www.newrelic.com/` (abgerufen am 15.11.2012)).

[180] vgl. Beck 1999, S. 50.

[181] vgl. Brown u. a. 2010.

[182] vgl. Seaman und Guo 2011, S. 27 f.

[183] vgl. Willke 2001, S. 86 f.

[184] vgl. Willke 2007, S. 111; Probst u. a. 2010, S. 195; Schneider 2001, S. 135 f.

[185] vgl. Fowler 2009.

[186] vgl. Brown u. a. 2010, S. 50.

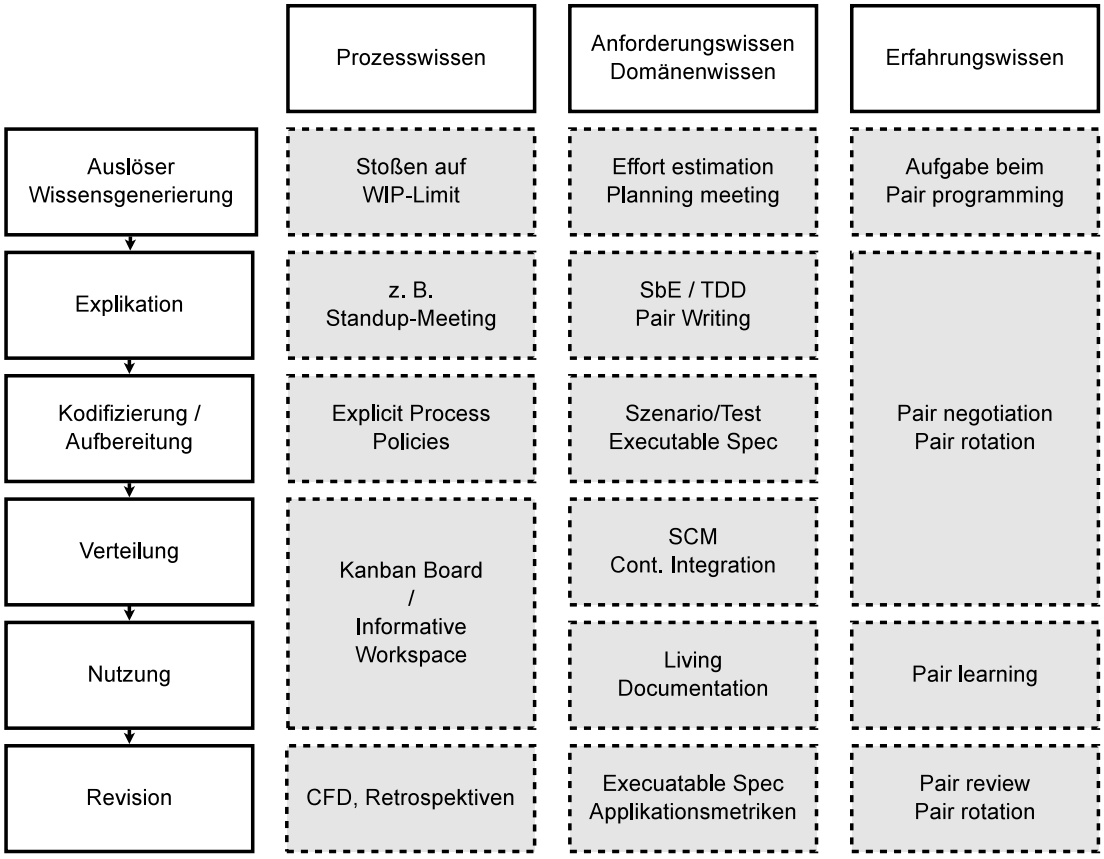

Abbildung 11: Wissenflüsse in agilen Teams

In der zweiten Spalte ist der Fluss des **Prozesswissens** skizziert: Eine Lernerfahrung kann z. B. durch das Stoßen auf ein WIP-Limit im Kanban Prozess angeregt werden, die Explikation bzw. Problembeschreibung erfolgt im Standup Meeting. Hier erarbeitet das Team eine Lösung für das Problem, bspw. durch Erhöhen des Limits oder Anpassung einer Prozess-Policy. Diese wird sofort auf dem Kanban-Board aktualisiert und ist somit verteilt. Die Revision des Wissens kann später in einer Retrospektive geschehen, aber auch durch CFD-Diagramme, welche eine Veränderung im Fluss (»Flow«) bemerken. Dies kann eine erneute Lernerfahrung auslösen, so dass der Prozess von neuem beginnt.

In der dritten Spalte von links wird der Fluss von **Anforderungswissen und/ oder Domänenwissen** gezeigt. Auslöser der Wissensgenerierung ist z. B. ein Planning meeting, in dem eine User story diskutiert wird. Im nächsten Schritt werden durch die SbE-Methode Akzeptanztests kollaborativ formuliert, in eine ausführbare Spezifikation überführt und die User story implementiert sowie

in das zentrale Versionskontrollsystem[187] und per CI integriert, d. h. der neue Quelltext inkl. der Spezifikation ist jetzt kodifiziert und verteilt im Sinne eines SCM-Prozesses. Aus der Spezifikation wird eine »Living Documentation«, d. h. das Wissen kann wieder von anderen Teammitgliedern oder auch Außenstehenden genutzt werden. Die Revision des Wissens kann auf mehrere Weisen passieren: Zum einen kann durch Lesen und Anpassen der Spezifikation die Anforderung verändert werden. Wird sie ohne Änderung der Implementierung verändert, so macht das CI-System darauf aufmerksam, dass die Implementierung von der Spezifikation abweicht. Die Auswertung Applikationsmetriken könnte ebenfalls zu einer direkten Rückkopplung über das (vermeintliche) Anforderungswissen führen, indem das Nutzungsverhalten sichtbar wird, und inwiefern die neu implementierte Softwareeigenschaft genutzt wird. Auch hier könnte eine neue Lernerfahrung in eine neue Iteration des Wissensmanagementprozesses münden.

Die rechte Spalte zeigt den Fluss von **Erfahrungswissen**. Ein Kreislauf könnte bspw. wie folgt aussehen: Ein Paarteil macht die Lernerfahrung, z. B. ein neu entdecktes Editorkommando, welches Zeit spart. Die Lernerfahrung wird direkt weitergegeben, indem der andere Paarteil diese direkt übernimmt. Wechseln Driver und Navigator, so kann das gelernte Wissen direkt vom anderen Partner angewandt werden (Pair learning). Durch Rotation von Paaren verteilt sich das Wissen über das neue Editorkommando langsam im Team. Weiterhin wird es dadurch auch ständig revidiert (Pair review): Ggf. kennt ein Teammitglied eine noch effektivere Tastenkombination, d. h. hier beginnt der Kreislauf erneut.

[187] Quelltext wird normalerweise in ein zentrales »version control«, wie z.B. Git (URL: `http://git-scm.com/` (abgerufen am 15.11.2012)) eingecheckt, d. h. verteilt, vgl. auch Abran u. a. 2004, 7–8

4.1.8 Zusammenfassung und kritische Betrachtung

In diesem Abschnitt wurden agile Methoden auf Wissensmanagementaktivitäten im Sinne des Wissensmanagements als Geschäftsprozess untersucht. Es konnte gezeigt werden, dass agile Methoden in den definierten Wissensarten den Prozess des Wissensmanagements unterstützen und durchlaufen.

Tabelle 11 zeigt eine nach Wissensarten differenzierte, zusammenfassende Darstellung der Wissensarten und der jeweils unterstützenden agilen Methoden.

Tabelle 11: Wissensarten und Unterstützung durch agile Methoden

Wissensart	Unterstützende agile Methoden
Domänenwissen	Crossfunktionale Teams, On-site Customer, SbE (Living Documentation), TDD, Planning Meeting / Effort Estimation, Pair Programming, Daily Meeting
Anforderungswissen	Crossfunktionale Teams, On-site Customer, SbE (Living Documentation), TDD, Planning Meeting / Effort Estimation, Pair Programming, Daily Meeting, CI, Applikationsmetriken
Applikationswissen	TDD, Coding standards, Planning Meeting / Effort Estimation, Pair Programming, Refactoring, Applikationsmetriken, Daily Meeting, CI, Infrastructure as Code
Prozesswissen	Kanban, Daily Meeting, Retrospektiven, Continuous Integration, Continuous Delivery
Erfahrungswissen	Crossfunktionale Teams, Pair Programming, Daily meeting, Planning Meeting / Effort Estimation, Retrospektiven

Agile Methoden unterstützen also in den definierten Wissensarten den Prozess des Wissensmanagement.

Es zeigt sich, dass crossfunktionale Teams und die kommunikativen Eigenschaften wie Daily und Planning Meeting, Pair Programming mehrere Wissensarten (jeweils vier) berühren, diese Methoden erscheinen daher als besonders effektiv.

Anzumerken ist, dass alle gezeigten Aktivitäten jeweils nur innerhalb des Systems »Team« aktiv sind, d. h. ein Wissensaustausch zwischen mehreren Teams findet durch den Einsatz agiler Methoden nicht statt.

Die bisherige Literatur (Chau u. a. 2003; Melnik und Maurer 2004) bezieht sich nur auf die Wissensverteilung von implizitem Wissen. Das vorgestellte Konzept erweitert und komplettiert die Sicht, indem es Explizierung, Dokumentation und Revision hinzufügt, und schließt somit den Wissenskreislauf.

An agilen Vorgehensweisen wird kritisiert, dass keine ausreichende Dokumentation erstellt wird[188]. Dies konnte hier teilweise widerlegt werden: Durch das Konzept der Living Documentation wird Domänenwissen und Anforderungswissen effektiv gespeichert. Agile Dokumentation[189] und Automatisierung (z. B. ausführbare Spezifikationen) als Methoden ergänzen dies. Dieses Vorgehen hat zwei weitere Vorteile: Dokumentation wird von Softwareentwicklern zumeist als störend und unproduktiv sowie schnell veraltet empfunden.[190] Dokumentation im Sinne von ausführbaren Spezifikationen kann nicht veralten, da sie durch CI-Systeme ständig validiert wird. Weiterführende Ideen und Ansätze zur Dokumentation in agilen Teams sind bei Stettina u. a. 2012 zu finden.

Kritisch betrachtet ist die Einordnung der agilen Methoden in die Prozessstufen mitunter nicht eindeutig zu leisten, denn eine agile Methode wirkt zumeist auf mehreren Stufen. Beispielsweise deckt die Spezifikation von Anforderungen mittels SbE die Generierung, Topographie, Explikation und Verteilung ab. Passiert die Anforderungsanalyse kollaborativ, also zwischen Kunden und Entwickler, so kann auch direkt eine Wissensrevision ausgelöst werden, indem der Entwickler kritische Nachfragen stellt. Dadurch werden beispielsweise fehlerhafte Spezifikationen früher erkannt, d. h. das Wissen muss nicht erst dann revidiert werden, wenn die Softwareeigenschaft schon implementiert und als fehlerhaft erkannt wurde. Diese Schwierigkeit bei der Systematisierung der Methoden ist allerdings zugleich ein Vorteil: Agile Methoden sorgen dafür, dass der Wissenskreislauf in sehr kurzen Zyklen durchlaufen wird, d. h. Wissen wird durch Kollaboration und Kommunikation ständig revidiert und hinterfragt.

Es sei hier noch einmal darauf hingewiesen, dass in dieser Studie nur ein Teilgebiet agiler Methoden, welches für Jimdo relevant ist, untersucht wurde. Eine Generalisierung der Ergebnisse ist also nicht ohne Weiteres möglich.

[188] vgl. Paetsch u. a. 2003, S. 5; vgl. Rubin und Rubin 2011, S. 130.
[189] vgl. Ambler 2012b.
[190] vgl. Chau u. a. 2003, S. 306; vgl. Stettina u. a. 2012, S. 31.

4.2 Überprüfung und Vervollständigung des Modells

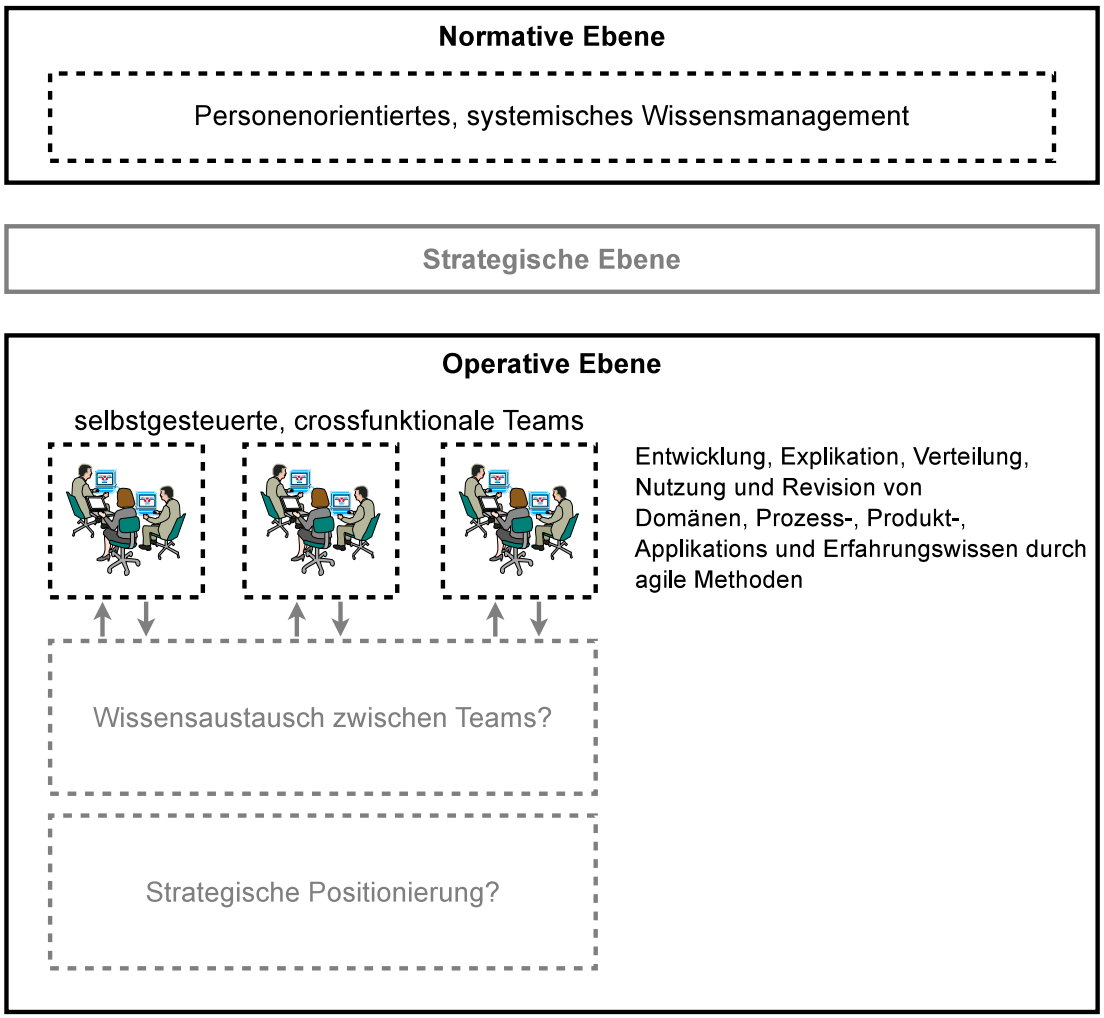

Abbildung 12: Einordnung agiler Methoden, Schritt 2

Im Folgenden wird ein Wissensmanagementmodell entwickelt, das auf agilen Methoden fußt und die bisherigen Erkenntnisse visualisieren soll. Im vorangegangen Abschnitt wurden agile Methoden in Bezug auf die Stufen des Wissensmanagements als Geschäftsprozess systematisiert. Es wurde festgestellt, dass durch agile Methoden ein ausreichendes Wissensmanagement innerhalb **crossfunktionaler** Teams sichergestellt werden kann. Aus diesem Grund wird das crossfunktionale Team als zentrale Methode in der agilen Vorgehensweise für die folgende Modellentwicklung herausgenommen. Zudem ist es bei Jimdo im Einsatz, Es wird in der Regel auch immer bei agilen Teams zugrunde

gelegt und ist somit als Grundvoraussetzung für agile Vorgehensweisen anzusehen[191].

Die Basis des Modells wurde mit Abbildung 6 entworfen. Dies zeigt die drei Ebenen des Managements in Anlehnung an das St. Galler Managementmodell. Auf normativer Ebene steht die Einordnung des Unternehmens in ein personenorientiertes, systemisches Wissensmanagement, wie in Kapitel 3 dargelegt wurde. Das Modell wird in Abbildung 12 weitergeführt. Auf operativer Ebene sind selbstgesteuerte, crossfunktionale Teams dargestellt, daneben der zusammenfassende Satz, dass durch agile Modelle die Entwicklung, Explikation, Verteilung, Nutzung und Revision von Domänen-, Prozess-, Anforderungs-, Applikations- und Erfahrungswissen realisiert werden können. Die graue Schrift symbolisiert bisher ungeklärte Aspekte, die über agile Methoden nicht abgedeckt werden, wie in den vorangegangen Abschnitten ausgeführt wurde. So nehmen sie keinen Einfluss auf die strategische Ausrichtung des Wissensmanagements, ein Wissensaustausch zwischen Teams[192] findet nicht statt, die Frage nach der strategischen Positionierung bleibt ebenfalls offen. In Bezug auf diese Aspekte besteht also Optimierungsbedarf für ein Wissensmanagementkonzept, wie es hier entwickelt werden soll.

An dieser Stelle kann die Leithypothese der vorliegenden Diplomarbeit überprüft werden:

Der Einsatz von agilen Softwareentwicklungsmethoden kann implizit für ein ausreichendes Wissensmanagement im Software Engineering sorgen, ohne explizit als strategisches Instrument des Wissensmanagements angewendet zu werden.

Vor dem Hintergrund der Ausführungen in dieser Arbeit, kann diese Hypothese also nur teilweise bestätigt werden. Der Einsatz agiler Methoden, wie er bei Jimdo stattfindet, kann auf operativer Ebene Wissensmanagement im Software Engineering hervorbringen, die strategische Ebene kann aber nicht bedient werden.

[191] »Self-organizing teams are at the heart of Agile software development« (Hoda u. a. 2011, S. 74)
[192] vgl. Chau u. a. 2003, S. 309.

Im Folgenden werden Maßnahmen und Ansätze vorgeschlagen, die das Modell vervollständigen können (siehe Abbildung 13), so dass es als Handlungsvorschlag und Arbeitsgrundlage für das Unternehmen Jimdo dienen kann. Auf strategischer Ebene werden vier Bereiche vorgeschlagen, die Anwendung finden sollten: Bestimmung der strategischen Positionierung anhand einer Wissenslandkarte, Personal- und Teamentwicklung, der strategische Einsatz von selbstgesteuerten, crossfunktionalen sowie (technisch) unabhängigen Teams und die Förderung von Communities of Practice. Die strategische Ebene beeinflusst über die Kontextsteuerung die Teams als Systeme auf der operativen Ebene.

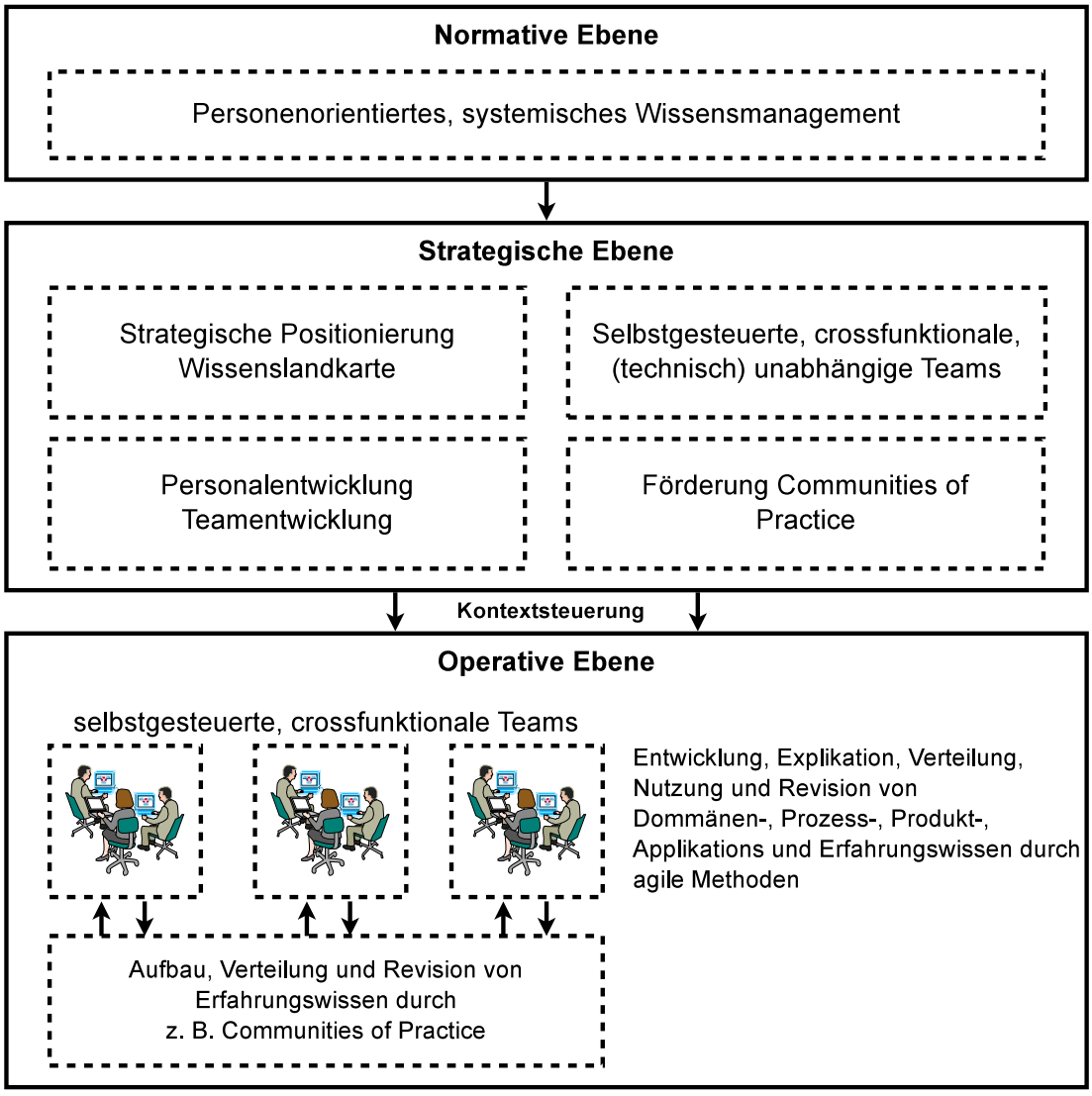

Abbildung 13: Vervollständigtes Wissensmanagementmodell, Schritt 3

Wissensmanagement wird bei Willke als Bestandteil des allgemeinen Managements angesehen. In seiner systemischen Sichtweise ist ein Unternehmen ein

komplexes dynamisches System. Es besteht »aus thematisch spezifischen und auf strategische Ziele ausgerichteten Kommunikationen«[193]. Die Steuerung des Systems kann immer nur in Selbst- und Kontextsteuerung bestehen.[194] Die Selbststeuerung bei Jimdo ist durch die selbstorganisierten Teams bereits sichergestellt. Die im hier entwickelten Managementmodell dargestellte Verbindung zwischen strategischer und operativer Ebene basiert somit auf der Kontextsteuerung. D. h. strategische Entscheidungen und Zielsetzungen betreffen Veränderungen des Kontextes bzw. der operativen Rahmenbedingungen, die das System Team hierdurch beeinflussen. In anderen Worten: Es wird auf strategischer Ebene das »Was« vorgegeben, aber nicht das »Wie«, dies wird durch Selbststeuerung auf der operativen Ebene realisiert. Dieser Grundgedanke ist bspw. auch bei Scrum und anderen agilen Methoden gegeben.

Die fehlenden Aspekte, die im Modell Abbildung 12 auf der operativen Ebene dargestellt sind, fehlender Wissensaustausch zwischen Teams sowie die Frage nach der strategischen Positionierung, könnten durch Maßnahmen auf der strategischen Ebene aufgefangen werden. Für das Vorgehen auf dieser Ebene wird zunächst eine strategische Positionierung vorgeschlagen, die den Blick auf Kernkompetenzen, Ressourcen und Wissensstrategien richtet. Als Instrument kann die Erstellung einer sog. Wissenslandkarte nach Willke[195] aufgegriffen werden. Das Unternehmen analysiert den Markt, in dem es agiert bzw. agieren will, seine Kundenstruktur und seine angestrebten Leistungen (Marktstrategie). Daneben richtet es den Blick nach innen und betrachtet welche Leistungen es mit den eigenen Ressourcen und Kompetenzen erreichen kann, welche Leistungen es erreichen will und welche Kompetenzen dafür benötigt werden (Produktstrategie). Das Unternehmen vergegenwärtigt sich auf diese Weise seine vorhandenen und benötigten Kompetenzen und Ressourcen und kann daraus Wissensstrategien und -ziele entwickeln. Aus der sichtbar gewordenen »Diskrepanz zwischen Wollen und Können«[196] können Programme der Personal- und Organisationsentwicklung abgeleitet werden.

Das Ermitteln von (fehlenden) Kernkompetenzen ist für den Erfolg von agilen Teams von großer Bedeutung, denn z. B. wird das Fehlen technischer Kom-

[193] Willke 2007, S. 106.
[194] vgl. Willke 2001, S. 92; vgl. Willke 2007, S. 107.
[195] vgl. ebd., S. 100.
[196] Ebd., S. 100.

petenzen als einer der Hauptgründe genannt, warum agile Teams scheitern.[197] Anhaltspunkte für die Bestimmung von Kernkompetenzen für das Software Engineering könnte SWEBOK geben. Beispielsweise könnten Teams daraus ableiten, welche Technologien eingesetzt werden sollten und welche ggf. eher eingekauft werden sollten, da sie nicht Bestand des Kerngeschäfts sind.

Weiterhin sind für eine sinnvolle Ergänzung des Wissensmanagementmodells der Einsatz von Maßnahmen zur Personalentwicklung sowie eine strategische Teamentwicklung empfehlenswert.[198]

Eine systemische Personalentwicklung kann durch den Einsatz von Methoden aus der systemischen Organisationsberatung erfolgen, welche zum Ziel haben neue Perspektiven zu eröffnen, die Selbstreflexivität und das Infragestellen der eigenen Position, von Überzeugungen und Beziehungen anzuregen.[199] Dazu könnten z. B. agile Coaches eingesetzt werden.[200]

Zur strategischen Teamentwicklung ist die adäquate Unterstützung selbstgesteuerter Teams von Bedeutung. Hoda u. a. 2011, S. 77 ff. stellen einige Maßnahmen bzw. Bedingungen dar:

- Die Unternehmenshierarchie sollte flach und die Unternehmenskultur offen und fehlertolerant sein.
- Mitarbeiter sollten nur einem Team angehören, ansonsten wird die Selbstorganisation erschwert und Teamkohäsion verringert.
- Bei der Einstellung neuer Mitarbeiter sollte besonders darauf geachtet werden, dass Personen eingestellt werden, die gut in agile Teams passen.
- Mentoren oder agile Coaches können die Teamentwicklung aktiv unterstützen, indem sie agile Methoden propagieren und vorleben.

---

[197] vgl. Silva u. a. 2008, S. 124.

[198] vgl. dazu Tabelle 1, Spalte »personenorientiertes Wissensmanagement«; Willke 2007, S. 81 sagt, »dass die Problematik der Schwächen und des fehlenden Wissens in das Thema des Lernens und der Personalentwicklung münden muss, so wie auf der Ebene der Organisation die Feststellung fehlender Kernkompetenzen das Thema der Organisationsentwicklung und des organisationalen Lernens aufruft«. Der Aspekt der Organisationsentwicklung soll zwar genannt, aber aus Platzgründen in dieser Diplomarbeit nicht weiter aufgegriffen werden.

[199] Beispiele für systemische Interventionswerkzeuge: Reflecting Team, Positive Konnotation, Rückspiegelungen, Reframing, Paradoxe Intervention, Fragen und Fragetechniken (z. B. Zirkuläre Fragen); vgl. Ellebracht u. a. 2009; vgl. Königswieser und Hillebrand 2005

[200] Dieses Thema soll an dieser Stelle nicht weiter vertieft werden, da es den Rahmen sprengen würde.

- Störende Teammitglieder, die z. B. mit der Selbstorganisation nicht zurechtkommen oder diese negativ beeinflussen, sollten aus Teams entfernt werden.

In Bezug auf selbstgesteuerte, crossfunktionale (technisch) unabhängige Teams, wie im Wissensmanagementmodell dargestellt, soll ein Aspekt betont werden. Für die Teamaufstellung ist - neben der Cross-funktionalität und Selbstorganisation - eine wichtige und kritische Voraussetzung, dass diese möglichst (technisch) unabhängig und entkoppelt von anderen Teams sein sollten, bspw. als sog. »Feature-Teams«, die ein Teilprodukt komplett selbst entwickeln und pflegen.[201] Bestehen Abhängigkeiten zwischen den Teams, so kann es sein, dass Applikations-, Domänen-, Anforderungs- und ggf. sogar Prozesswissen ausgetauscht werden müssen. Dies erhöht die Komplexität des Wissensflusses, und die Selbstorganisation ist dadurch eingeschränkt, da der Koordinations- und Kommunikationsaufwand steigt.[202] Je entkoppelter Teams arbeiten können, desto besser funktioniert das Wissensmanagement innerhalb der Teams. Beispielsweise wäre es unökonomisch, wenn eine zu implementierende User Story des einen Teams auf noch zu entwickelnde Grundlagen des anderen Teams aufsetzt. Die Folge wäre, dass Wissen über Teams verteilt und aktuell gehalten werden müssten. An dieser Stelle sei darauf hingewiesen, dass ein Wissensaustausch zwischen den Teams essentiell und erwünscht ist. Die eben beschriebene Entkopplung bezieht sich auf die Vermeidung von »lähmenden« Abhängigkeiten auf technischer und der Prozessebene.

Während Applikations-, Domänen-, Anforderungs- und Prozesswissen innerhalb crossfunktionaler (Feature-)Teams gut aufgehoben sind und nicht weiter verteilt werden müssen, so sind die Erfahrungen[203], die die Teams oder ihre Mitglieder machen, auch potentiell für andere Mitglieder der Organisation interessant. Hier sollte also ein weiterer Kreislauf bzw. Geschäftsprozess des Wissens für Erfahrungen eingebracht werden. Es bieten sich dafür z. B. Communities of Practice an, die schon teilweise im Unternehmen Jimdo im Einsatz sind.

Laut North[204] sind solche Wissensgemeinschaften Personengruppen, die an einem gemeinsamen Thema Interesse haben und gemeinsam Wissen aufbauen

[201] vgl. Poppendieck und Poppendieck 2010, S. 132 f.
[202] vgl. ebd., S. 133.
[203] Zum Beispiel die Einführung eines neuen Programms zur Softwareentwicklung
[204] vgl. North 2011, S. 163.

und austauschen wollen. Diese Gruppen bestehen über einen längeren Zeitraum, die Teilnahme ist persönlich und freiwillig. Der systemische Gedanke der Selbstorganisation und der agile Gedanke der persönlichen Anwesenheit sind in CoP also erfüllt.

Ein weiterer Kreislauf bzw. Geschäftsprozess des Wissens in Bezug auf Erfahrungswissen über den Einsatz von CoP, der auf operativer Ebene den Wissenskreislauf zwischen Teams ermöglicht, könnte wie folgt aussehen:

Verankert in der strategischen Vorgehensweise des Wissensmanagements ist, dass Mitarbeiter aktiv dazu ermutigt werden (**Auslöser der Wissensgenerierung**), eigene Wissensgemeinschaften zu bilden.[205] Anreize zur **Wissensteilung** und zum **Wissenserwerb** können durch »wissensbezogene Kriterien in Mitarbeiterbeurteilung und -entwicklung« gesetzt werden. Mitarbeiter können gefragt werden, was sie im letzten Jahr getan haben, um ihre Kompetenzen zu steigern oder wie sie Kollegen und der Organisation in der Weiterentwicklung der Wissensbasis geholfen haben.[206] Oder auch, welchen Wissensgemeinschaften sie beiwohnen. Die **Dokumentation** und **Revision** von Erfahrungswissen könnte innerhalb von CoP wiederum im bestehenden Unternehmenswiki in Form eines Mikroartikels von Willke geschehen. So wäre ein Wissenskreislauf zwischen Teams vollständig skizziert.

Das entwickelte Konzept ist (durch die Auswahl der agilen Modelle) nur bedingt unternehmensspezifisch und hätte somit das Potential, auch in ähnlichen Unternehmen derselben Branche und ähnlicher Unternehmenskultur adaptiert zu werden. Es wäre also eine Erweiterung des personenorientierten Wissensmanagements zu einem **»agilen Wissensmanagement«**.

4.3 Zusammenfassung und kritische Betrachtung

Ergebnis dieses Hauptkapitels ist ein Konzept für ein Wissensmanagementmodell, das auf agilen Methoden basiert, wie sie bei Jimdo zum Einsatz kommen. Das Modell stellt einen Vorschlag dar, wie das Wissensmanagement, welches durch die agilen Methoden bereits implizit existiert, durch weitere Maßnahmen ergänzt werden könnte.

[205] vgl. North 2011, S. 168.
[206] vgl. ebd., S. 160.

Zunächst wurden agile Methoden, die bei Jimdo praktiziert werden, daraufhin untersucht, inwiefern sie Wissensmanagement unterstützten. Dazu wurden sie vom Autor in Anlehnung an das Modell des Wissensmanagements als Geschäftsprozess von Willke systematisiert. Das Ergebnis war, dass die untersuchten agilen Methoden in den definierten Wissensarten (Domänen-, Anforderungs-, Prozess-, Applikations- und Erfahrungswissen) den Prozess des Wissensmanagements unterstützen und dass durch agile Methoden ein ausreichendes Wissensmanagement innerhalb crossfunktionaler Teams sichergestellt werden kann. Jedoch wurde auch verdeutlicht, dass die Wissensflüsse nur jeweils innerhalb des Systems »Team« aktiv sind, d. h. ein Wissensaustausch zwischen mehreren Teams durch den Einsatz agiler Methoden nicht stattfindet.

Im Anschluss an die Analyse der agilen Methoden wurde die Leithypothese überprüft. Sie besagt, dass durch agile Methoden ein ausreichendes Wissensmanagement sichergestellt wäre. Dies konnte für Wissen innerhalb crossfunktionaler Teams bestätigt werden. Darüber hinaus wurde festgestellt, dass sich durch den Einsatz agiler Methoden noch kein komplettes Wissensmanagementkonzept ergibt. Dafür bedarf es weiterer Überlegungen und zusätzlicher Maßnahmen auf der strategischen Ebene des Managements. Als strategische Instrumente wurden das Bestimmen einer strategischen Positionierung des Unternehmens mittels einer Wissenslandkarte, Team- und Personalentwicklungsmaßnahmen, Förderung von selbstgesteuerten, crossfunktionalen, unabhängigen Teams sowie von CoP vorgeschlagen. Steuerung wurde im Sinne von Kontextsteuerung betont. Das Konzept für ein Wissensmanagement wurde in einem Modell dargestellt und es wurden kurz Vorschläge zur Umsetzung ausgeführt. Es konnte ein Kreislauf für Erfahrungswissen skizziert werden, der die Kommunikation zwischen Teams ermöglicht, welcher durch den bloßen Einsatz von agilen Teams nicht gegeben ist.

Das entwickelte Modell ist als Handlungsvorschlag für das Unternehmen zu verstehen. Die Verbesserungsvorschläge sollten nicht als Toolbox angesehen werden, sondern durch »gut platzierte Pilotprojekte« ausprobiert werden, ob sie sich in der Praxis bewähren.[207] Dies folgt dem agilen Grundgedanken Jimdos.

Zur kritischen Betrachtung der Vorgehensweise und Ergebnisse ist zunächst darauf hinzuweisen, dass der Fokus im Rahmen einer Diplomarbeit stark ein-

[207] vgl. Willke 2007, S. 67.

gegrenzt werden muss, denn Wissensmanagement als Querschnittsdisziplin betrifft das ganze Unternehmen auf verschiedenen Ebenen. Es schließen sich bspw. Themen und Maßnahmen der Organisations-, Team- und Personalentwicklung an, z. B. über die Konzepte des personalen und organisationalen Lernens. Diese Themen und eigenständige Gebiete konnten in dieser Arbeit nur knapp angerissen werden.

Die folgenden Aspekte wurden im Kapitel behandelt, sollten aber für die Übertragung in die Praxis kritisch weitergedacht werden. Die vielen kommunikativen Anteile von agilen Methoden, wie Daily Meeting, Retrospektiven, Pair Programming, Effort estimation (meeting) sind besonders effizient, um Wissen im Team aktiv und aktuell zu halten, ohne dass aufwändige Dokumentationen geschrieben werden. Prozesswissen wird mittels Kanban entwickelt, dokumentiert, verteilt und revidiert. Auf der technischen Seite sorgen neuere Methoden wie Live-Applikationsmetriken für eine Revision von Anforderungswissen. Durch Automatisierung und ausführbare Spezifikationen wird Wissen so dokumentiert bzw. kodifiziert, dass es sich selbst validiert und »revidiert«. Vorteilhaft ist, dass diese Dokumentation automatisiert stattfindet und keine zusätzliche Arbeitskraft bindet und dass automatisierte Test eine zusätzliche Sicherheit zu manuellen Tests bieten. Dennoch sollte kritisch hinterfragt werden, inwiefern sich komplett auf automatisierte Tests verlassen werden kann oder sollte.[208]

Es wurde gezeigt, dass crossfunktionale Teams ein essentieller Baustein im Wissensmanagement in agilen Teams sind. Hier sollte also strategisch ein besonderes Augenmaß drauf gelegt werden, dass Teams mit den richtigen Kompetenzen »ausgestattet« sind (Kernkompetenzen, Wissenslandkarte). Auch sollte die Personalentwicklung darauf eingestellt werden, dass die einzelnen Teammitglieder »T-shape skills« haben bzw. entwickeln, um einen möglichst breiten Horizont zu erlangen.

Der Wissenserwerb, also wie das Unternehmen durch seine Mitglieder zu neuem Wissen gelangt und dieses filtert (»information overload«), ist in dem entwickelten Modell nicht explizit geregelt. Schneider schlägt vor, »Menschen in Mustererkennung, Problemlösung und Selbstbescheidung (Loslassen-Können) zu erziehen«[209], d. h. es wird Mitarbeitern nicht vorgeschrieben, wie

[208] vgl. Haugset und Stalhane 2012, S. 5296 f.
[209] Schneider 2001, S. 94 f.

sie zu neuem Wissen gelangen sollen, sondern ihnen wird durch unterstützende Maßnahmen indirekt dabei geholfen.

Es ist nicht zu vernachlässigen, dass Agilität und Selbstorganisation den Teammitgliedern große Disziplin abverlangen, Ambler und Lines[210] nennen bspw. folgende Anforderungen:

- Andauernde Kommunikation und Kollaboration mit dem Team (z. B. Pair Programming) und dem Kunden (z. B. SbE) ist schwieriger und ggf. unangenehmer als isoliertes Arbeiten an einem isoliertem Arbeitsplatz.
- Das Treffen, Respektieren und Einhalten kollektiver Teamentscheidungen in selbstorganisierten Teams ist schwerer als die Vorgaben eines klassischen Projektmanagers zu befolgen.
- Lebendige Dokumentation und ausführbare Spezifikationen mittels Test-first Development aktuell zu halten, erfordert große Disziplin der Softwareentwickler.
- Dokumentation, die nicht als ausführbare Spezifikation vorliegt, z. B. Software-architektur- und Systemübersichten oder Benutzeranteilungen müssen erstellt und gepflegt werden.

Agile Coaches könnten als Mentoren die Teams dabei unterstützen, agile Praktiken diszipliniert durchzuhalten und Probleme offen anzusprechen.[211] Sie könnten z. B. dazu beitragen, dass die Prinzipien der »agilen Dokumentation«[212] kontinuierlich beachtet werden, um dem Problem von fehlender oder veralteter Dokumentation entgegenzuwirken.

Weiterhin ist kritisch anzumerken, dass die agilen Methoden unterschiedlich schwer einführbar sind. Während Pair Programming sofort umgesetzt werden kann (und wie gezeigt werden konnte, auf den Wissenskreislauf direkt einen großen Effekt hat), braucht das richtige Aufstellen von crossfunktionalen Teams, die wirklich alle Kompetenzen und Befugnisse besitzen, sicherlich längere Zeit. Zudem sind Methoden wie TDD im Nachhinein, also in Verbindung mit einer bestehenden Software, schwieriger einzuführen als bei neu gestarteten Projekten.[213] Auch die Einführung automatisierter Akzeptanztests ist

[210] Ambler und Lines 2012, S. 484 ff.
[211] vgl. Hoda u. a. 2010, S. 293.
[212] vgl. z. B. Ambler 2012b; vgl. Rüping 2003.
[213] vgl. Breivold u. a. 2010, S. 35.

mit größeren Anfangskosten verbunden.[214] Bravo und Goldman weisen darauf hin, dass das Erlernen von agilen Softwareentwicklungsmethoden wie TDD und Refactoring ein Prozess ist, der nicht durch das Lesen von Büchern passieren kann, sondern nur durch Training. Sie schlagen Trainingssessions, sog. »Coding Dojos« vor, um die Adaption zu verbessern.[215]

Hier wird eine Gemeinsamkeit von systemischem Wissensmanagement und agiler Softwareentwicklung sichtbar, die beide keine kurzfristigen Rezepte bieten, sondern auf langsame und kontinuierliche Verbesserung und Nachhaltigkeit angelegt sind.[216]

Tabelle 12 zeigt in einer zusammenfassende Übersicht als Ergebnis dieses Kapitels.[217]

Tabelle 12: Agiles Wissensmanagement in der Übersicht

Fokus	**Aktivitäten**	**Werkzeuge**	**Probleme**
- Kommunizieren - Kooperieren - Automatisieren	- Verbinden - Vernetzen - Entwickeln - Agile Dokumentation, Ausführbare Spezifikationen	- Meetings, informelle Treffen, Mentorenprogramme - Teamentwicklung und Personalentwicklung - Agile Methoden wie Daily Meetings, Retrospektiven, Pair Programming, TDD, SbE etc. - Communities of Practice	- Kultur des Teilens und der Zusammenarbeit, Verstehen - Einführungskosten - Disziplin in Bezug auf Selbstorganisation und Werkzeugverwendung - Entkopplung von Teams - Eigenständiger Wissenserwerb

[214] vgl. Haugset und Stalhane 2012, S. 5296.
[215] vgl. Bravo und Goldman 2010.
[216] »Geduldige Stärkung der Wissensbasis« (Willke 2007, S. 72 ff.)
[217] in Anlehnung an den personenorientierten Zugang von Tabelle 1

5 Fazit

Im Verlauf der vorliegenden Arbeit wurde Wissensmanagement als integraler Bestandteil des Software Engineerings untersucht. Anhand des Beispiels des Web 2.0-Unternehmens Jimdo und den dort eingesetzten agilen Modellen und Methoden wurde ein Wissensmanagementmodell für das Unternehmen konzipiert.

Zu Beginn die Leithypothese dieser Diplomarbeit formuliert, die zum Teil bestätigt werden konnte:

> **Der Einsatz von agilen Softwareentwicklungsmethoden kann implizit für ein ausreichendes Wissensmanagement im Software Engineering sorgen, ohne explizit als strategisches Instrument des Wissensmanagements angewendet zu werden.**

Daraus ergaben sich folgende Leitfragen:

Warum ist Wissensmanagement im Software Engineering wichtig?

Software Engineering ist eine wissensintensive Tätigkeit, welche verschiedene Wissenstypen (implizites sowie explizites) verlangt und den Fluss von relevanten Wissensarten (Domänen-, Prozess-, Anforderungs-, Applikations- und Erfahrungswissen) voraussetzt. Als Grundlage wurde festgehalten, dass der Zweck von Wissensmanagement darin besteht, die zukünftige Innovationskompetenz durch kollektives Lernen und kontinuierliche Revision des eigenen Wissens zu unterstützen.

Wie sehen bisherige Ansätze in der Literatur von Wissensmanagement im Software Engineering aus?

Bisherige Ansätze, die Wissensmanagement im Software Engineering betrachten, sind wenig vorhanden und behandeln agile Methoden kaum. In der Literatur ist zumeist eine Differenzierung zwischen den beiden Extremen personenorientiertes und dokumentenorientiertes Wissensmanagement zu finden. Agile Softwareentwicklung kann nach dieser Einteilung dem personenorientierten Wissensmanagementansatz zugeteilt werden.

Sind bisherige Ansätze bei Jimdo einsetzbar?

Bisherige Ansätze von Wissensmanagement im Software Engineering beinhalten größtenteils Zusammenstellungen von Werkzeugen und sind im dokumen-

tenorientierten Zugang verortet. Für das Unternehmen Jimdo wurde aufgezeigt, dass seine Arbeitsweise einem personenorientierten, systemischen Wissensmanagementansatz zugeordnet werden kann, denn es folgt bereits einer agilen Vorgehensweise und einer dementsprechenden Unternehmenskultur, die stark auf Selbstorganisation setzt.

Wo unterstützen agile Methoden Wissensmanagement? Wie kann dies untersucht werden?

In Anlehnung an das Modell des Wissensmanagements als Geschäftsprozess von Willke konnte aufgezeigt werden, das agile Methoden - in der bei Jimdo verwendeten Kombination - die definierten Wissensarten durchlaufen. Eine zentrale Erkenntnis dabei war, dass agile Methoden Wissensmanagement auf operativer Ebene unterstützen. Eine weitere wichtige Erkenntnis ist, dass agile Methoden alleine oder durch Kombination den kompletten Kreislauf des Wissensmanagement durchlaufen und Wissen somit nicht nur verteilt und gespeichert, sondern auch regelmäßig revidiert wird. Formen der Automatisierung wie ausführbare Spezifikationen unterstützen aktiv das Wissensmanagement, da sie sich selbst validieren und revidieren.

Wo sind Grenzen agiler Methoden im Wissensmanagement? Wo müssen traditionelle Methoden des Wissensmanagement ergänzend ansetzen?

Auf operativer Ebene und in Bezug auf einzelne Teams unterstützen agile Methoden Wissensmanagement, allerdings ist der Wissensaustausch zwischen Teams nicht explizit geregelt. Agile Modelle machen zudem keine explizite Aussage zur strategischen Positionierung eines Unternehmens.

Wie könnte eine Vorgehensweise für ein Wissensmanagement als integraler Bestandteil für Software Teams bei Jimdo skizziert aussehen?

Auf normativer Ebene könnte für Jimdo ein personenorientiertes, systemisches Wissensmanagement festgelegt werden. Die strategische Ebene könnte durch die strategische Positionierung, die Stärkung crossfunktionaler (technisch) unabhängiger Teams und verschiedene Methoden der Personal- und Teamentwicklung ausgestaltet werden.

Der Wissensaustausch, in Form von Aufbau, Verteilung und Revision von Erfahrungswissen zwischen Teams könnte ergänzend über die Förderung von selbstorganisierten CoPs initialisiert werden.

Folgende **kritische Faktoren bei der Einführung des Modells bzw. potentielle Probleme** wurden identifiziert:

- Es ist große Disziplin der Teams und Mitglieder in Bezug auf Selbstorganisation und Werkzeugverwendung nötig.
- Am besten funktioniert das Modell, wenn Teams untereinander (technisch) entkoppelt sind.
- Es können initiale Aufwände wie z. B. für das Setup von automatisierten Akzeptanztests entstehen.
- Teams und Mitglieder müssen in eigenständiger Weise neues Wissen erwerben, filtern und in den Unternehmenskontext einbauen.

In Hinblick auf die Vorgehensweise in dieser Arbeit ist auf eine Schwierigkeit der Untersuchung hinzuweisen. Diese besteht darin, dass die agilen Methoden nicht immer eindeutig auf eine Stufe des Wissensmanagements als Geschäftsprozess eingeordnet werden konnten, da diese oft mehrere Stufen gleichzeitig durchlaufen. Dies wird allerdings dadurch aufgefangen, dass immer der Schritt der Revision des Wissens existiert, so dass der Kreislauf des Wissensmanagement gewährleistet ist.

Im Rahmen einer Diplomarbeit können nicht alle Aspekte ausführlich behandelt werden, die das Thema berühren. Als Ausblick auf mögliche weiterführende Untersuchungen von Wissensmanagement im Software Engineering und agilen Methoden kann zusammengefasst werden:

- Der Blick auf Lernkonzepte, die hinter agilen Ansätzen stehen, z. B. im Hintergrund der »Lernenden Organisation« von Senge u. a. 2011.
- Die Betrachtung von Formen des Nicht-Wissens, z. B. bei Schneider 2006.
- Eine Ausweitung der Untersuchung auf z. B. Lean Software Development Methoden, auch unter dem Blickwinkel der Vermeidung der Generierung unnötigen Wissens in Form von »Waste«.
- Der Fokus auf Softwarearchitektur und die Absicherung architekturellen Wissens.
- Die Frage der Wissensverteilung »Wer muss was wissen?« in Bezug auf selbstgesteuerte unabhängige Teams und Komplexitätsreduzierung.

Letztendlich sollte sich Wissensmanagement in einem agilen Softwareunternehmen - im Sinne des Agilen Manifests - daran messen, inwiefern die entwickelte Software wartungs- und weiterentwicklungsfähig bleibt, um die zukünftige Innovationskompetenz nachhaltig zu gewährleisten.

Literatur

Abran, Alain u. a. (2004). *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE.

Adzic, G. (2011). *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications.

Allspaw, J. und J. Robbins (2010). *Web Operations: Keeping the Data On Time*. Sebastopol: O'Reilly Media, Incorporated.

Ambler, Scott (2012a). *Agile/Lean Documentation: Strategies for Agile Software Development*. URL: `http://www.agilemodeling.com/essays/agileDocumentation.htm` (abgerufen am 13. 10. 2012).

Ambler, Scott (2012b). *Best Practices for Agile/Lean Documentation*. URL: `http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm` (abgerufen am 13. 10. 2012).

Ambler, S.W. und M. Lines (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. Boston: Pearson Education.

Anderson, D.J. (2010). *Kanban*. Sequim: Blue Hole Press.

Babar, M.A. u. a. (2009). *Software Architecture Knowledge Management: Theory and Practice*. Berlin Heidelberg: Springer.

Basili, V. u. a. (1994). »Experience factory«. In: *Encyclopedia of Software Engineering*. Hrsg. von J. J. Marciniak. New York: John Wiley & Sons, S. 469–476.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. The XP Series. Boston: Addison-Wesley.

Beck, K. und C. Andres (2005). *Extreme Programming Explained: Embrace Change*. 2. Aufl. The XP Series. Boston: Addison-Wesley.

Bjørnson, Finn Olav und Torgeir Dingsøyr (Okt. 2008). »Knowledge management in software engineering: A systematic review of studied concepts, fin-

dings and research methods used«. In: *Inf. Softw. Technol.* 50.11, S. 1055–1068.

Bravo, Mariana und Alfredo Goldman (2010). »Reinforcing the Learning of Agile Practices Using Coding Dojos«. In: *Agile Processes in Software Engineering and Extreme Programming*. Hrsg. von Alberto Sillitti u. a. Bd. 48. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, S. 379–380.

Breivold, H.P. u. a. (2010). »What Does Research Say about Agile and Architecture?« In: *Software Engineering Advances (ICSEA), 2010 Fifth International Conference on*, S. 32 –37.

Brown, N. u. a. (2010). »Managing technical debt in software-reliant systems«. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, S. 47–52.

Bryant, Sallyann u. a. (2006). »The Collaborative Nature of Pair Programming«. In: *Extreme Programming and Agile Processes in Software Engineering*. Hrsg. von Pekka Abrahamsson u. a. Bd. 4044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, S. 53–64.

Chau, T. u. a. (2003). »Knowledge sharing: agile methods vs. Tayloristic methods«. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, S. 302–307.

Cohn, M. (2006). *Agile Estimating And Planning*. Robert C. Martin Series. Upper Saddle River: Prentice Hall Professional Technical Reference.

Cook, Nigel u. a. (2012). »Cloud management«. In: *Journal of Internet Services and Applications* 3 (1), S. 67–75.

Dybå, T. und T. Dingsøyr (2008). »Empirical studies of agile software development: A systematic review«. In: *Information and software technology* 50.9, S. 833–859.

Earl, M. (2001). »Knowledge management strategies: towards a taxonomy«. In: *Journal of Management Information Systems* 18 (1), S. 215–233.

Ellebracht, H. u. a. (2009). *Systemische Organisations- und Unternehmensberatung: Praxishandbuch für Berater und Führungskräfte*. Wiesbaden: Gabler.

Fernandes, J.M. und M. Almeida (2010). »Classification and Comparison of Agile Methods«. In: *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the*, S. 391 –396.

Fowler, Martin (Juni 2008). *Agile Versus Lean*. URL: `http://martinfowler.com/bliki/AgileVersusLean.html` (abgerufen am 14. 10. 2009).

Fowler, Martin (Feb. 2009). *Technical Debt Quadrant*. URL: `http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html` (abgerufen am 14. 10. 2009).

Haugset, B. und T. Stalhane (2012). »Automated Acceptance Testing as an Agile Requirements Engineering Practice«. In: *System Science (HICSS), 2012 45th Hawaii International Conference on*, S. 5289 –5298.

Hoda, R. u. a. (2010). »Organizing self-organizing teams«. In: *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*. Bd. 1, S. 285 –294.

Hoda, Rashina u. a. (2011). »Supporting Self-organizing Agile Teams«. In: *Agile Processes in Software Engineering and Extreme Programming*. Hrsg. von Alberto Sillitti u. a. Bd. 77. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, S. 73–87.

Humble, J. und D. Farley (2010). *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. The Addison-Wesley Signature Series. Boston: Addison-Wesley.

Königswieser, R. und M. Hillebrand (2005). *Einführung in die systemische Organisationsberatung*. Heidelberg: Carl-Auer-Systeme-Verlag.

Levy, M. und O. Hazzan (2009). »Knowledge management in practice: The case of agile software development«. In: *Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on*, S. 60 –65.

Lindvall, M. u. a. (2004). »Agile software development in large organizations«. In: *Computer* 37.12, S. 26 –34.

Loukides, M. (Juni 2012). *What is DevOps?* URL: http://radar.oreilly.com/2012/06/what-is-devops.html (abgerufen am 01. 11. 2012).

Melnik, G. und F. Maurer (2004). »Direct verbal communication as a catalyst of agile knowledge sharing«. In: *Agile Development Conference, 2004*, S. 21–31.

Müller, Barbara (2009). *Wissen managen in formal organisierten Sozialsystemen*. Wiesbaden: Gabler.

Navarro, Antonio (2009). »A SWEBOK-based Viewpoint of the Web Engineering Discipline«. In: *Journal of Universal Computer Science* 15.17, S. 3169–3200.

Nonaka, I. und H. Takeuchi (1986). »The New New Product Development Game«. In: *Harvard Business Review* January–February.

Nonaka, I. und H. Takeuchi (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. New York: Oxford University Press, USA.

North, K. (2011). *Wissensorientierte Unternehmensführung: Wertschöpfung durch Wissen*. 5. Aufl. Wiesbaden: Gabler Verlag.

Paetsch, F. u. a. (2003). »Requirements engineering and agile software development«. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, S. 308 –313.

Paul, D. (2007). *Continuous Integration*. Boston: Pearson Education.

Poppendieck, M. und T. Poppendieck (2010). *Leading Lean Software Development: Results Are Not the Point*. 2. Aufl. Addison-Wesley Signature Series. Boston: Addison-Wesley.

Probst, G. u. a. (2010). *Wissen managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen*. 6. Aufl. Wiesbaden: Gabler Verlag.

Rubin, Eran und Hillel Rubin (2011). »Supporting agile software development through active documentation«. In: *Requirements Engineering* 16 (2), S. 117–132.

Rubin, K.S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison Wesley Signature Series. Boston: Addison Wesley Professional.

Rüping, A. (2003). *Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects*. Wiley Software Patterns Series. West Sussex: Wiley.

Rus, I. und M. Lindvall (2002). »Knowledge management in software engineering«. In: *Software, IEEE* 19.3, S. 26 –38.

Schneider, K. (2009). *Experience and Knowledge Management in Software Engineering*. Berlin Heidelberg: Springer.

Schneider, U. (2001). *Die sieben Todsünden im Wissensmanagement*. Frankfurt am Main: Frankfurter Allg. Buch.

Schneider, U. (2006). *Das Management der Ignoranz*. Wiesbaden: Deutscher Universitäts-Verlag GWV Fachverlage GmbH.

Schwaber, K. (2009). *Agile Project Management with Scrum*. Redmond: Microsoft Press.

Seaman, Carolyn B. und Yuepu Guo (2011). »Measuring and Monitoring Technical Debt«. In: *Advances in Computers* 82, S. 25–46.

Senge, P.M. u. a. (2011). *Die fünfte Disziplin: Kunst und Praxis der lernenden Organisation*. Systemisches Management. Schäffer-Poeschel Verlag.

Silva, Liana u. a. (2008). »Applying XP to an Agile–Inexperienced Software Development Team«. In: *Agile Processes in Software Engineering and Extreme Programming*. Hrsg. von Pekka Abrahamsson u. a. Bd. 9. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, S. 114–126.

Stettina, C.J. u. a. (2012). »Documentation Work in Agile Teams: The Role of Documentation Formalism in Achieving a Sustainable Practice«. In: *Agile Conference (AGILE), 2012*, S. 31 –40.

Williams, L. und R.R. Kessler (2003). *Pair Programming Illuminated*. Boston: Addison-Wesley.

Willis, John (Juli 2010). *What Devops Means to Me*. URL: http://www.opscode.com/blog/2010/07/16/what-devops-means-to-me/ (abgerufen am 13. 10. 2012).

Willke, H. (2001). *Systemisches Wissensmanagement*. Uni-Taschenbücher S. Stuttgart: Lucius & Lucius.

Willke, H. (2007). *Einführung in das systemische Wissensmanagement*. 2. Aufl. Heidelberg: Auer.

Eidesstattliche Erklärung

Ich versichere, dass ich die beiliegende Diplomarbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe.

_____ _____

(Ort, Datum) (Unterschrift)