Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften

Bachelorarbeit

# Werkzeugunterstützung von agilen Prozessen in der Softwareentwicklung

**Dennis Keitzel**

6keitzel@informatik.uni-hamburg.de
Studiengang Informatik
Matr.-Nr. 5859720

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Heinz Züllighoven |
| Zweitgutachter: | Prof. Dr. Horst Oberquelle |
| Betreuer: | Dipl.-Wirt.Inf. Jörg Rathlev |

# Inhaltsverzeichnis

# 1 Einleitung

Softwaretechnik beschreibt eine Disziplin in der Informatik, die sich nach BALZERT (vgl. [Bal09]) aus den Bereichen Softwareentwicklung, Softwaremanagement und Softwarequalitätsmanagement zusammensetzt. Er definiert das Gebiet als die *"Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen."*[1]. Wenn umfangreiche Software entwickelt werden soll, kann man sich dieser Prinzipien, Methoden und Werkzeuge bedienen. Sie versprechen jeweils eine bessere Beherrschbarkeit für die unterschiedlichen Teilgebiete der Softwareentwicklung.

Für das strukturierte Entwickeln von Software hat die Softwaretechnik vielfältige Vorgehensweisen herausgearbeitet, die angewandt werden können. Neben klassischen Varianten, wie dem Wasserfallmodell, werden derzeit vermehrt agile Varianten eingesetzt. Im Gegensatz zu dem starren, nichtiterativen Wasserfallmodell will die agile Softwareentwicklung den Entwicklungsprozess flexibler gestalten.

Einige agile Methoden werden üblicherweise in Abwesenheit von Computern verwendet. Für die Anforderungsanalyse werden beispielsweise Softwarespezifikationen zusammen mit dem Kunden auf sogenannten Story-Cards notiert. Dies entspricht dem agilen Prinzip möglichst kundennah zu entwickeln. Neben dem Arbeiten auf Papier besteht für die Umsetzung der agilen Softwareentwicklung aber auch die Möglichkeit einer Unterstützung durch Software. Diese verspricht eine bessere Handhabung hinsichtlich der jeweiligen agilen Methoden oder dem Gesamtprozess.

## 1.1 Fragestellung & Ziel

Derzeit sind einige unterstützende Werkzeuge für die Softwareentwicklung auf dem Markt erhältlich. Das Spektrum reicht von Produkten für einen ganz speziellen Bereich in der Softwareentwicklung, bis hin zu Lösungen, die eine komplette Integration der Entwicklung anstreben. Angeboten werden freie Varianten, aber auch kommerzielle und kostenpflichtige. Die Problematik bei der Auswahl besteht darin, fundiert feststellen zu können, welches Produkt für einen bestehenden Entwicklungsprozess am besten geeignet ist. Dazu muss das Werkzeug im Vorfeld bewertet werden. Anhand einer solchen Bewertung lassen sich Vor- und Nachteile erkennen, sowie unterschiedliche Werkzeuge miteinander vergleichen.

Für die Durchführung einer Bewertung stellt das Gebiet der Evaluationforschung einige Methodiken bereit. Eine Softwareevaluation verfolgt nach HOLZINGER (vgl. [Hol03]) im allgemeinen drei Hauptziele. Sie soll Probleme der Benutzer im Umgang mit der Soft-

[1] Seite 17 in [Bal09]

ware ermitteln, die Funktionalität der Software bewerten und die Effekte einer Software auf den Benutzer bestimmen. Im Rahmen dieser Arbeit soll festgestellt werden, welche genauen Methoden der Evaluation für die Bewertung einsetzbar sind und anschließend erläutert werden, warum die Verwendung eines Kriterienkatalogs sinnvoll ist.

Ziel der Arbeit ist es, solch einen Kriterienkatalog zusammenzustellen und anschließend exemplarisch zwei Werkzeuge anhand des Kriterienkatalogs zu bewerten. Es wird ausführlich dargelegt, wie der Kriterienkatalog ensteht und welche Kriterien enthalten sind. Um eine Übertragbarkeit auf ähnliche Situationen zu gewährleisten, wird insbesondere die generelle Herangehensweise an Fragestellungen dieser Art eingehend dargestellt.

## 1.2 Fallbeispiel

Das Zusammenstellen von Kriterien für die Bewertung der Werkzeuge wird unter anderem auf Basis von theoretischen Überlegungen der Wissenschaft, wie zum Beispiel aus dem Bereich der Usability[2] erfolgen, aber vor allem anhand empirischer Daten eines Fallbeispiels. Die Erhebung von Kriterien anhand eines Fallbeispiels erlaubt dem Ansatz praxinäher zu sein, als wenn die Kriterien allein aus theoretischen Grundüberlegungen entstehen würden. Zu beachten ist, dass der auf diese Art und Weise entstehende Kriterienkatalog auf genau dieses Fallbeispiel ausgerichtet wird. Das eigentliche Ziel dieser Arbeit ist aber den Weg aufzuzeigen, wie Kriterien aus einem Fallbeispiel entnommen und für eine Bewertung genutzt werden können.

Das Fallbeispiel ist ein Unternehmen, welches nach einer Selbstbeschreibung *"Lösungen aus den Bereichen Consulting, Software-Entwicklung sowie Informations- und Kommunikationstechnologie im Umfeld web-orientierter Anwendungen und Technologien"* anbietet. Es hat sich darauf spezialisiert, Altsysteme durch moderne Lösungen zu ersetzen. Die Softwareentwicklung findet agil in einem Team von unter zehn Personen statt. Es werden typische agile Werte als Basis gesetzt und agile Prinzipien und Methoden angewendet. Die genaue Definition der Werte, Prinzipien und Methoden wird im folgenden Kapitel 2 zur Terminologie erfolgen. Wichtig ist aber, dass agile Softwareentwicklung einen entscheidenen Einfluss auf den Entwicklungsprozess nimmt und daher signifikant ist für eine Unterstützung durch ein Werkzeug.

Das Fallbeispiel ist indirekt auch Ausgangspunkt für die Aufgabenstellung in dieser Arbeit. Es wird bereits ein Werkzeug zu Unterstützung eingesetzt, doch zeigen sich einige Schwächen in der täglichen Benutzung. Es wird vermutet, dass die Effizienz der Arbeit durch ein besser geeignetes Werkzeug steigt.

An dieser Stelle möchte ich mich herzlichst für die freundliche und kompetente Unterstützung seitens der Mitarbeiter bedanken. Die Zusammenarbeit fand stets in einer angenehmen wie produktiven Atmosphäre statt. Sie haben dieser Arbeit ermöglicht, einen

[2] Englischsprachiger Begriff für die Gebrauchstauglichkeit eines Produkts

praxisrelevanten Bezug herzustellen, indem Kriterien direkt aus dem Umfeld tatsächlich angewendeter agiler Methoden extrahiert werden konnten.

## 1.3 Gliederung

Die Arbeit wird sich zunächst mit der grundsätzlichen Terminologie aus dem Bereich der agilen Softwareentwicklung beschäftigen. Sollten die Begriffe zum Großteil bereits bekannt sein, kann dieses Kapitel übersprungen und nur bei Bedarf darauf zurückgegriffen werden. Wurde diese Grundlage geschaffen, wird sich dem Vorgehen gewidmet die Fragestellung zu beantworten, indem wissenschaftliche Methoden zur Bewertung von Software herangezogen werden. Ist das Vorgehen geklärt, wird anschließend der eigentliche Kriterienkatalog zusammengestellt. Dabei werden drei Klassen von Kriterien unterschieden und zwar fachliche und technische Kriterien, sowie Kriterien aus dem Bereich der Usability, die jeweils auf unterschiedliche Weise ermittelt werden. Anschließend werden anhand des zusammengestellten Kriterienkatalogs zwei Werkzeuge exemplarisch bewertet. Zuletzt wird im Fazit das Ergebnis der Arbeit zusammengefasst und es soll reflektiert werden, welche Aussagekraft der Kriterienkatalog besitzt.

# 2 Terminologie

Dieses Kapitel beschreibt - alphabetisch sortiert - Begriffe aus dem Gebiet der agilen Softwareentwicklung, die im Laufe der Ausführung der Arbeit verwendet werden. Die Beschreibungen basieren zum Teil auf der Literatur, aber auch auf der Basis von Informationen aus dem Fallbeispiel. Wie in der Einleitung bereits erwähnt, kann dieses Kapitel übersprungen und nur bei Bedarf genutzt werden.

## 2.1 Begriffe

| **Begriff** | **Beschreibung** |
|---|---|
| 10-Minute-Build | Teil der Continuous Integration. Ein 10-Minute-Build baut die gesamte Anwendung und lässt alle Tests innerhalb von zehn Minuten durchlaufen. Nach BECK (vgl. [BA04]) erhält der Entwickler dadurch die schnelle Möglichkeit für ein Feedback. |
| Agile Werte | Bilden das Fundament der agilen Softwareentwicklung. Sie wurden 2001 bei einem Treffen von 17 Autoren aus dem Gebiet der agilen Softwareentwicklung als Agiles Manifest formuliert und von HIGHSMITH und FOWLER (vgl. [HF01]) in einem Artikel veröffentlich. Sie lauten im Original:<br>• Individuals and interactions over processes and tools.<br>• Working software over comprehensive documentation.<br>• Customer collaboration over contract negotiation.<br>• Responding to change over following a plan. |

Agile Prinzipien

Agile Prinzipien basieren auf den agilen Werten und bilden Handlungsgrundlagen in Form von Leitsätzen für die agile Arbeit. BECK (vgl. [BA04]) hat einige dieser Leitsätze formuliert, wovon ein paar exemplarisch aufgeführt werden:

- *"Humanity"*: Es sind Menschen, welche Software entwickeln. Sie benötigen ein Gefühl der Sicherheit, müssen Bestätigung erfahren, sollen sich weiterbilden können und brauchen einen gewissen Grad an Vertrautheit im Umgang mit anderen.
- *"Mutual Benefit"*: Jede Aufgabe soll zur Zufriedenheit aller beteiligten Personen beitragen. Das betrifft zum Beispiel den aktuellen und den zukünftigen Entwickler, sowie den Kunden.
- *"Self-Similarity"*: Die Struktur für Lösungen von Problemen lassen sich oft an anderen Stellen wiederverwenden. Das betrifft nicht nur Quelltexte, sondern auch Vorgehen.
- *"Improvement"*: Perfektion ist in der Softwareentwicklung kaum erreichbar. Der beste Weg ist, Praktiken und Designs inkrementell zu verbessern.
- *"Reflection"*: Es soll nicht einfach nur gearbeitet werden, sondern es soll in regelmäßigen Abständen reflektiert werden, wie und warum gearbeitet wird.
- *"Quality"*: Qualität ist keine Variable, die die Geschwindigkeit einer Entwicklung angibt. In der Regel werden hochqualitative Arbeiten schneller erledigt, als wenn versucht wird die Qualität künstlich zu erniedrigen, um ein schnelleres Ergebnis zu erhalten.
- *"Baby Steps"*: Der Overhead für große Änderungen ist durch kleine Schritte kleiner, als wenn die Änderung in einem großen Schritt durchgeführt wird.

| | |
|---|---|
| Agile Methoden | Methoden, um die agile Softwareentwicklung umzusetzen. Einige von BECK (vgl. [BA04]) vorgestellten Methoden sind:<br>• *"Sit Together"*: Die Entwicklung soll in einem ausreichend großen Raum stattfinden, in dem alle Entwickler Platz haben.<br>• *"Energized Work"*: Es soll nur solange gearbeitet werden, wie die jeweilige Person auch produktiv und konzentriert arbeiten kann. Sich überarbeiten und Überstunden machen, wirkt sich negativ auf die kommenden Tage und damit auch negativ auf das gesamte Team aus.<br>• *"Pair Programming"*: Das Programmieren im Paar. Für die genaue Erläuterung, siehe Seite 8.<br>• *"Test-First Programming"*: Zunächst wird ein Test auf den zu ändernen Quelltext geschrieben, der zunächst fehlschlägt. Erst danach wird der Quelltext entsprechend geändert, bis der Test schließlich durchläuft.<br>• *"Story Cards"*: Das Verwenden von Story-Cards. Für die genaue Erläuterung, siehe Seite 9. |
| Backend | Eine Anwendung kann in Schichten aufgeteilt werden. Dabei bezeichnet ein Frontend das Interface zum Benutzer, während das Backend im Hintergrund Daten halten oder auch bearbeiten kann. |
| Bug & Defect | Ein Bug bezeichnet das Fehlerverhalten eines Programms zum Zeitpunkt der Entwicklung. Ein Defect unterscheidet sich zum Bug nur dadurch, dass ein Defect zum Zeitpunkt des produktiven Einsatz auftritt. |
| Bugtracking | Eine Anwendung, die das Erfassen, Dokumentieren und Verfolgen von Bugs unterstützt. |
| Build | Der Prozess, in dem die Anwendung gebaut wird. |
| Buildserver | Ein Server, auf dem die Anwendung gebaut wird. |

| | |
|---|---|
| Burn-Down-Chart | Eine grafische Repräsentation, welche die Beziehung zwischen der noch verbleibenden Arbeit zur verbleibenden Zeit ausdrückt. |
| Continuous Integration | Entwickler sollen frühzeitig und oft Änderungen am Quelltext in die Versionsverwaltung einchecken. Continuous Integration beschreibt den Prozess des regelmäßigen Neubauens und Testen der Anwendung. Dadurch werden Probleme schnell aufgedeckt und die Integration von Änderungen erleichert. [1] |
| Entwicklungsprozess | Übernimmt die Steuerung der Softwareentwicklung anhand von festgelegten Vorgehensmodellen. |
| Forum | Virtueller Ort für asynchrone Diskussionen. Die Beiträge sind in der Regel für alle Benutzer und auf dauer einsehbar. |
| Integrationstest | Testet das korrekte Zusammenarbeiten verschiedener, voneinander abhängiger Komponenten. |
| Iteration | Bezeichnet einen Entwicklungszyklus. Er kann, abhängig vom Vorgehensmodell, eine Planung, eine Analyse, einen Entwurf, eine Implementation, sowie das Testen einer Anwendung enthalten.[2] |
| Pair Programming | Zwei Personen sitzen an einem Computer und programmieren, analysieren, designen und testen eine Anwendung gemeinsam. Nach BECK (vgl. [BA04]) steigt dadurch die Qualität der Software. |
| Pool | Konzept für einen Ort, in dem Stories gesammelt werden können. |
| Release | Bezeichnet die fertige Version eine Software, die in den produktiven Einsatz übergehen kann. |

[1] Für mehr Informationen, siehe [BA04]
[2] Für mehr Informationen, siehe [BA04]

| | |
|---|---|
| SCM | SCM steht für "Source-Code-Management", der englische Begriff für "Versionsverwaltung". Es ist ein System für das Erfassen von Änderungen an Dateien. Dadurch lassen sich ältere Versionen von Dateien wiederherstellen. Zudem lässt es sich von mehreren Personen so nutzen, dass alle Entwickler den selben Versionsstand der Quelltexte haben. |
| Schätzpoker | Methode um den Aufwand von Aufgaben zu schätzen. Mithilfe eines Kartendecks, das Karten mit Aufwandspunkten enthält, werden Aufgaben von mehreren Entwicklern gleichzeitig geschätzt, indem jeder eine Karte für die Aufgabe angibt. Sollten starke Unteschiede auftreten, wird darüber diskutiert und erneut geschätzt, bis eine Einigung erzielt wird. |
| Selenium | Ein freies Testframework, um automatisiert Webanwendungen zu testen. |
| Spikes | Ein Spike bezeichnet die Zeit, in der sich ein Entwickler über einen unbekannten Sachverhalt, wie beispielsweise eine neue Technologie, informieren kann. |
| Story & Story-Cards | Eine Story beschreibt eine Funktionsanforderung an die Anwendung. Festgehalten werden diese auf Story-Cards. Stories werden in der Regel in ihrem Aufwand geschätzt. [3] |
| Task | Eine Story kann in Tasks unterteilt werden. Tasks beschreiben dann einzelne Bearbeitungsschritte, um die Story zu erfüllen. |
| Truck-Factor | Der Truck-Factor beschreibt die Anzahl der Leute, die von einem Truck überfahren werden können, bevor das Projekt scheitern könnte. |
| Unittest | Das Testen kleiner Einheiten von Quelltexten, bei objektorientierter Programmierung; in der Regel einzelne Methoden. |
| Usecase | Ein Usecase, englisch für "Anwendungsfall", besteht aus zusammenhängenden Aufgaben, die von einem Akteur durchgeführt werden, um ein Ziel zu erreichen. |

[3] Für mehr Informationen, siehe [BA04]

| | |
|---|---|
| User-Acceptance-Test | Ein Test einer Anwendung durch den Kunden. Es wird geprüft, ob das Verhalten der Anwendung der Spezifikation entspricht, indem zum Beispiel Grenzwerte bei der Dateneingabe verwendet werden. |
| Werkzeug | Im Sinne dieser Arbeit eine softwaretechnische Unterstützung der agilen Softwareentwicklung. |
| Wiki | Ein Hypertext-System für das Anzeigen und Bearbeiten von Inhalten im Browser. |

# 3 Vorgehen

Es existieren zahlreiche Methoden um ein Thema wissenschaftlich aufzuarbeiten. In diesem Kapitel wird begründet dargelegt, welche Methodiken verwendet und darüber hinaus wie diese im Detail angewendet werden. Ausgangspunkt für die Betrachtung des Vorgehens ist das zu erreichende Ziel. Wie in der Einleitung dargelegt, ist dies die Bewertung von zwei Werkzeugen zur Unterstützung agiler Prozesse.

Die Bewertung einer Software kann mittels einer Evaluation durchgeführt werden. HOLZINGER (vgl. [Hol03]) unterscheidet prinzipiell zwischen zwei Arten von Bewertungen wenn Software evaluiert werden soll; die *formative* sowie die *summative* Evaluation. Sie unterscheiden sich durch den zeitlichen Aspekt, bezogen auf den Software-Lifecycle. Während die formative Evaluation zum Zeitpunkt der Softwareentwicklung stattfindet und diesen Prozess mithilfe von qualitativen Interviews und Beobachtungen laufend unterstützt, bewertet die summative Evaluation ein bereits bestehendes Softwareprodukt. Diese Arbeit beschränkt sich auf die Untersuchung von bereits bestehenden Software-Produkten, wodurch die Wahl der Bewertungsart zwangsweise auf die summative Methode fällt.

Einzelne Kriterien einer Evaluation werden nach Holzinger anhand von Evaluationsmitteln beurteilt. OPPERMANN (vgl. [OMRK92]) unterscheidet diese Evaluationsmittel anhand der Einbezogenheit der Menschen, welche die Software benutzen. Er grenzt drei Kategorien voneinander ab:

**Subjektive Evaluationsmittel**

Die einzelnen Kriterien werden von den Nutzern selbst mithilfe drei verschiedener Methode beurteilt:

- Mündliche Befragung: Den Nutzern werden Fragen gestellt, die zur Beurteilung der gewählten Kriterien dienlich sind.
- Ein Fragebogen wird erstellt und von den Nutzern ausgefüllt.
- Die Nutzer werden aufgefordert, während einer Aufgabenbewältigung ihre Überlegungen, Probleme und ihre jeweiligen Handlungsalternativen laut vor sich her zu sagen.

**Objektive Evaluationsmittel**

Subjektive Einflüsse der Nutzer, wie Emotionen, Vorlieben und Vorurteile werden vermieden. Unterschieden wird anhand der An- und Abwesenheit des Beobachters:

- Anwesende Beobachtung: Der Beobachter sitzt neben dem Nutzer und versucht anhand der beobachteten Handlungen, Fehler, Ausführungszeiten (Messung) und anderer wahrgenommener Attribute, die Evaluationskriterien entsprechend zu beurteilen.
- Abwesende Beobachtung: Hierbei wird der Nutzer, zum Beispiel durch eine Videoaufzeichnung, indirekt beobachtet.

**Leitfadenorientierte Evaluationsmittel**

Das fertige Softwareprodukt wird möglichst objektiv anhand eines Prüfleitfadens beurteilt. Der Prüfleitfaden ergibt sich aus den typischen Aufgaben des Softwaresystems. Während der Aufgabenbewältigung wird entsprechend der Evaluationskriterien beurteilt. Für leitfadenorientierte Evaluationsmittel ist kein Nutzer notwendig.

Um diese Bewertung der zwei Werkzeuge transparent und nachvollziehbar durchführen zu können, wird in dieser Arbeit die leitfadenorientierte, summative Evaluation ohne Anwesenheit von den eigentlichen Nutzer verwendet. Ein Kriterienkatalog ermöglicht diese Art der Evaluation. Im Folgenden wird zunächst der Begriff des Kriterienkatalogs definiert und anschließend werden, ausgehend vom Ziel, die einzelnen Schritte abgeleitet, die für die Erstellung notwendig sind.

## 3.1 Kriterienkatalog

Unter einem Kriterienkatalog ist in dieser Arbeit eine Sammlung von relevanten Kriterien zur quantitativen Bewertung und zum Vergleichen von Werkzeugen zur Unterstützung von agilen Prozessen in der Softwareentwicklung zu verstehen. Laut BAUMGARTNER (vgl. [Bau97]) bietet ein Kriterienkatalog im Allgemeinen folgende Vorteile:

1. Das Verfahren ist kostengünstig umzusetzen, da eine fachkundige Person, sowie eine Kopie der Anwendung, die auf einem Rechner läuft, genügt.
2. Die Durchführung ist methodisch sauber. Durch die schrittweise, immer gleiche Abarbeitung der Kriterienliste bleibt das Verfahren objektiv. Dadurch lassen sich Ergebnisse leicht miteinander vergleichen.
3. Die Organisation der Evaluation ist einfach, da die Software unabhängig von ihrem eigentlichen Einsatzort getestet werden kann.

Baumgartner gibt jedoch auch folgende Probleme zu bedenken:

1. Es ist nur schwer zu beurteilen, ob ein Kriterienkatalog vollständig ist oder ob der gewählte Detailierungsgrad ausreichend ist. Eventuell müssen laufend neue Kriterien mit aufgenommen werden.

2. Durch die Losgelöstheit vom eigentlichen Einsatzort stellt sich die Frage, ob die erlangten Ergebnisse überhaupt aussagekräftig genug für eine spätere Anwendung der Software sind.

3. Fehlende oder strittige Bewertungsverfahren und Gewichtungen für Kriterien können dazu führen, dass derjenige, der die Software evaluiert, seine subjektive Meinung zu stark mit in die Bewertung einfließen lässt. Sollte dies nicht im Vorfeld eindeutig festgelegt sein, so ist das Verfahren nicht mehr objektiv.

Die aufgezählten Probleme lassen sich im Kontext dieser Ausarbeitung allerdings teilweise entkräften. Durch die Aufnahme von Kriterien, die speziell auf die Bedürfnisse der späteren Einsatzumgebung zugeschnitten sind, fällt die Frage nach der Aussagekraft in dieser Hinsicht weg. Dies erlaubt im Umkehrschluss allerdings auch eine geringere Übertragbarkeit auf andere Umgebungen, da der Kriterienkatalog so speziellere Anforderung stellt, die woanders eventuell nicht relevant sind. Das Ziel ist aber nicht, einen für alle Interessierten allgemeingültigen Kriterienkatalog zu erstellen, sondern einen Weg aufzuzeigen, wie dieser für die eigene Umgebung erstellt werden kann.

Wie im folgendem Abschnitt dargelegt wird, werden fehlende Bewertungsverfahren vermieden und sollten daher nicht auftreten. Gewichtungen von Kriterien werden zunächst offen gelassen und müssen später, wenn der Kriterienkatalog zusammengestellt wurde, mit den zukünftigen Nutzern zusammen vorgenommen werden. Zu beachten ist dabei, dass der Kriterienkatalog dadurch noch deutlicher auf eine spezielle Umgebung ausgerichtet wird.

Ein optimaler Kriterienkatalog erfüllt nach MEIER (vgl. [Mei95]) neun Voraussetzungen, die nach Möglichkeit eingehalten werden sollten. Mit Erhöhung der zutreffenden Voraussetzungen steigt auch die Aussagekraft eines angewendeten Kriterienkatalogs. Meier hat diese Voraussetzungen im Kontext von Lernsoftware erstellt, daher wird der Kriterienkatalog für diese Arbeit um zwei, hier nicht relevante Voraussetzungen, gekürzt.

1. Die Beurteilung erfolgt durch einen - im Idealfall geschulten - Fachmann.

2. Die Kriterien sind vollständig, gültig und zuverlässig.

3. Die Kriterien liegen in strukturierter Form vor und sind in Kriteriumskategorien (mit Überschriften) unterteilt.

4. Die Kriteriumskategorien sind einzeln als Prüfinstrumente anwendbar.

5. Die Kriterien sind sachlich korrekt, verständlich, treffend und knapp formuliert.

6. Der Katalog ist einfach und im Idealfall elektronisch auswertbar. Voraussetzung hierfür ist unter anderem, dass die Kriterien in Sachzusammenhänge vorstrukturiert und die Bewertungsskalen einfach konstruiert sind (z.B. Ja/Nein-Antworten).

7. Der Kriterienkatalog ist sprachlich und grafisch einwandfrei und übersichtlich zu bearbeiten.

Ob die Voraussetzungen aber tatsächlich eingehalten werden, lässt sich nur schwer beurteilen. Das Einhalten einer strukturierten Form lässt sich vergleichsweise leicht umsetzen und überprüfen, aber wie kann sichergestellt werden, dass die Kriterien vollständig sind? Diese Frage kann an dieser Stelle nicht beantwortet werden. Sie wird erneut im Rahmen der Bewertung aufgegriffen.

### 3.1.1 Systematik

Die Architektur des Kriterienkatalogs und des Vorgehens wird sich an folgende Systematik halten, die im Kern von FRANKE ET AL. (vgl. [FMP08]) stammt:

- Der Kriterienkatalog besteht aus einzelnen Kriterien, die jeweils im Detail beschrieben werden.
- Ein Kriterium ist ein Merkmal, welches Einfluss auf die Qualität der Software hat. Dieser Einfluss wird in einer Motivation erläutert.
- Indikatoren definieren, wie Merkmale überprüft werden können.
- Anhand eines Bewertungsschemas ergibt sich die Bewertung.

### 3.1.2 Bewertungsverfahren

Das Bewertungsverfahren definiert das Zuweisen der Beobachtungen zu Werten. HOLZINGER (vgl. [Hol03]) unterscheidet dabei vier Methoden:

- Grading (Einstufung): Die Evaluanden werden anhand eines vordefinierten Beurteilungsmaßstabes eingestuft.
- Ranking (Reihung): Für eine Reihung werden die Evaluanden zueinander in Relation gesetzt und beurteilt. Es entsteht eine Ordinalskala mit einer Aussage wie gut - besser - am besten, die jedoch keine Aussage zu den Abständen untereinander macht.
- Scoring (Punktevergabe): Im Unterschied zum Ranking werden hier Punkte vergeben, deren Abstände untereinander gleich und "bedeutungsvoll" sind. Nur mit dieser Methode sind summative Operationen wie Addition oder Division zulässig.
- Apportioning (Aufteilung, Zuteilung): Entsprechend der Wertigkeit eines Evaluanden werden vorhandene Ressourcen aufgeteilt (als Beispiel kann die Aufteilung eines Staatshaushaltes genommen werden).

Auch das Bewertungsverfahren orientiert sich in dieser Arbeit an der Vorgehensweise von FRANKE ET AL.. Dort erfolgt die Bewertung der einzelnen Kriterien durch eine Zuordnung der Beobachtungen zu einer der Kategorien: *voll erfüllt*, *überwiegend erfüllt*, *teilweise erfüllt* und *nicht erfüllt*. Diese Kategorien werden zusätzlich mit Punkten von 3 für *voll erfüllt* bis 0 für *nicht erfüllt* belegt. Die Verteilung der Beobachtungen auf die vier Kategorien muss für jedes Kriterium einzeln erfolgen. Bei der Bewertung von Kandidaten darf von dem Schema, in begründeten Ausnahmefällen, abgewichen werden.

Dieses Verfahren ist in Anbetracht der vorgestellten Methoden von Holzinger eine Komposition aus der Einstufungs- und Punktevergabemethode. Holzinger schlägt außerdem eine Bewertung anhand von Schulnoten von 1 bis 5 vor. Durch die hier gewählte Art der Bewertung werden allerdings summative Operationen erleichtert. Dies betrifft beispielsweise, wie im nächsten Abschnitt vorgestellt wird, das Gewichten von Ergebnissen.

### 3.1.3 Gewichtung

Die angestrebte Bewertung zweier Werkzeuge in dieser Arbeit soll möglichst unabhängig von Subjektivität sein. Steht der mögliche Einsatz von solchen Werkzeugen aber bevor und sollen mögliche Kandidaten miteinander verglichen werden, so können persönliche Einschätzungen und Vorlieben eine wichtige Rolle spielen. Ebenso können Kriterien unterschiedlich relevant sein. Daher bietet das gewählte Bewertungsverfahren, wie ALARCÓN (vgl. [Ala06]) es ebenfalls ausgeführt hat, die Möglichkeit Kriterien zu Gewichten. Aufgrund der Nutzung der Punktevergabe-Methode, ist eine einfache Art und Weise eine Gewichtung durchzuführen, die Multiplikation der erreichten Punktzahl mit einem Multiplikator. Als Beispiel könnte es vier Stufen der Priorisierung geben, wobei $x$ jeweils für die erreichte Punktezahl steht und der Zahlenwert für die Priorisierung:

- Besonders wichtig: $x * 2$
- Keine Priorisierung: $x * 1$
- Weniger wichtig: $x * 0.5$
- Nicht relevant: $x * 0$

Wird für eine Kriterium der Wert 2 erreicht, kann durch eine Gewichtung der Wert beispielsweise verdoppelt werden, falls das Kriterium besonders wichtig ist, oder aber halbiert werden, wenn das Kriterium weniger wichtig ist. Sollte ein Kriterium für einen Vergleich nicht relevant sein, kann es durch die Priorisierung *"Nicht relevant"* aus der Bewertung herausgenommen werden.

In dieser Arbeit wird allerdings keine Gewichtung vorgenommen werden, da Vorlieben und persönliche Einschätzungen der späteren Nutzer nur unzureichend bekannt sind. Es wäre aber ohne weiteres möglich, nachträglich noch eine Gewichtung zu berücksichtigen.

## 3.2 Kriterien

Um die Menge der Kriterien überschaubar zu halten, sollen diese in verschiedene Klassen eingeteilt werden. In diesem Kapitel wird dargelegt, welche Klassen von Kriterien sich aus dem Umfeld der Werkzeugunterstützung für agile Prozesse ergeben. Zu jeder Klasse wird außerdem gezeigt, wie die einzelnen Kriterien der jeweiligen Klasse gefunden werden können. Beispielsweise lassen sich Kriterien zur Gebrauchstauglichkeit direkt aus der *ISO 9241, Teil 110 Grundsätze der Dialoggestaltung* ableiten, wohingegen sich Kriterien zu Mindestanforderungen an die Software aus dem Kontext der Einsatzumgebung ergeben. Damit wird auch ein Unterscheidungsmerkmal klar. Eine Klasse teilt sich für die zugehörigen Kriterien die Methode der Kriteriengenerierung.

Im Folgenden wird näher auf die drei Klassen eingegangen, die für die Bewertungskriterien in dieser Arbeit gewählt wurden.

### 3.2.1 Fachliche Kriterien

Fachliche Kriterien sind solche, die relevant für den Entwicklungsprozess selber sind. In meinen Fall stammen diese aus dem Kontext des Fallbeispiels. Die Gewinnung von Kriterien erfolgt in zwei Schritten.

**Schritt 1:** Erstellen von Szenarien, die Abläufe im Entwicklungsprozess beschreiben.

Im Fallbeispiel existieren unterschiedliche Perspektiven auf den Entwicklungsprozess. Diese sind durch einen eigenen Aufgabenbereich und die damit verbundenen unterschiedlichen Tätigkeiten im Einzelnen charakterisiert. Die zwei wichtigsten Perspektiven im Fallbeispiel werden durch die Rollen des Entwicklers und des Beraters[1] definiert. Neben diesen beiden Rollen wurde noch die Rolle des Kunden identifiziert. Dieser nimmt am agilen Entwicklungsprozess einen großen Stellenwert ein, bleibt hinsichtlich einer Werkzeugunterstützung aber eher im Hintergrund. Daher wird der Kunde im Zusammenhang mit dem Berater erwähnt, da dieser im direkten Kontakt mit dem Kunden steht.

Jede Rolle nimmt eigene Positionen während der Entwicklung ein, weswegen sie zunächst auch getrennt betrachtet werden müssen. Dies bedeutet allerdings nicht, dass sie nicht auch gemeinsam betrachtet werden können, schließlich arbeiten Entwickler und Berater auch zusammen. Allerdings wird jede Rolle durch eigene Abläufe im Unternehmen charakterisiert. Diese Abläufe gilt es in Form von Szenarien zu extrahieren.

Um an die benötigten Informationen, also die Abläufe während der Entwicklung, zu gelangen, wird die Methodik der Befragung gewählt. Befragt werden jeweils ein erfahrener Entwickler und ein Berater. Da lediglich vorhandenes Wissen abgefragt wird, reicht es jeweils eine Person in einem offenem Interview zu befragen. Das Befragen von mehreren Personen würde keine neuen Szenarien ergeben. Zur Sicherheit wird das jeweilige

[1] Der englischsprachige Begriff *Consultant* ist ebenso geläufig

Ergebnis der Interviews aber nochmal von einer anderen Person verifiziert. Die Ergebnisse der Interviews sind jeweils Szenarien aus Sicht des Entwicklers und des Beraters. Da die Rolle des Kunden nicht direkt mit dem Kunden selbst geklärt werden konnte, basieren dessen Informationen auf Erfahrungswerte der Entwickler und Berater. Durch den ständigen Kundenkontakt können einige wichtige Aspekte festgehalten werden.

**Schritt 2:** Extrahieren von Kriterien aus den Szenarien

Ein Szenario kann mehrere Schritte beinhalten. Dabei muss jeder Schritt durch das zu bewertende Werkzeug nach Möglichkeit unterstützt werden, sofern dies sinnvoll erscheint. Aus jedem Schritt soll ein Kriterium abgeleitet werden. Wird beispielsweise in einem vereinfachten Szenario eine Story abgeschlossen, so bildet genau dieser Vorgang ein Kriterium: Stories müssen abgeschlossen werden können. Bedacht werden muss dabei, dass für jedes Kriterium zusätzlich zur Beschreibung, der Indikator an dem beobachtet werden soll definiert, sowie ein Bewertungsschema festgelegt werden muss. Das Ergebnis sind die im Detail beschriebene Kriterien.

### 3.2.2 Technische Kriterien

Technische Kriterien sind solche, die sich aus der Umgebung der Software ergeben, in der sie eingesetzt wird. Viele lassen sich aus bereits bestehenden Kriterienkatalogen mit Bezug zu Software aus der Literatur entnehmen. Vor allem im Bereich der Lernsoftware sind bereits einige Kriterienkataloge erstellt und auch miteinander verglichen[2] worden. Zusätzlich werden weitere Kriterien mittels explorativen Anwendens des bereits genutzten Softwarewerkzeugs aus dem Fallbeispiel anhand von technischen Eigenschaften gewonnen. Da die Ermittlung der technischen Kriterien eine qualitative Methode ist, wird hinsichtlich der Menge der Kriterien kein Anspruch auf Vollständigkeit gewährleistet. Der Anspruch ist aber, zumindest alle wichtigen Kriterien zu enthalten.

### 3.2.3 Usability-Kriterien

Usability, oder auf deutsch *Gebrauchstauglichkeit*, ist ein nach DIN EN ISO 9241 Teil 11 definierter Begriff, der Anforderungen an die Effektivität der Lösung einer Aufgabe, die Effizienz der Handhabung des Systems und an die Zufriedenheit der Nutzer einer Software stellt. Dazu beschreibt der Teil 110 der DIN EN ISO 9241 einige Grundsätze für die Interaktion zwischen Benutzer und System.

Aus diesen Grundsätzen sollen Kriterien für die Bewertung entwickelt werden. Dazu bietet sich die von NIELSEN (vgl. [Nie93]) vorgestellte Methode der heuristischen Evaluation an, welche Usability-Probleme bei der Benutzung von Software anhand einer im

[2] Ein großer Vergleich zu bestehenden Kriterienkatalogen aus dem Gebiet der Lernsoftware findet sich in [Ala06]

Vorfeld definierten Sammlung von Heuristiken identifiziert. Dazu untersucht eine kleine Gruppe von Evaluatoren die Software und überprüft die Einhaltung der Heuristiken. NIELSEN hat allerdings beobachtet, dass die Anzahl der gefundenen Usability-Probleme stark von der Anzahl der Evaluatoren (Abbildung 3.1) abhängt. Ein ausführliches Testen der Usability liegt aber außerhalb des Skopus dieser Arbeit. Es soll daher ausreichend sein, eine grobe Vergleichbarkeit zwischen den Werkzeugen herzustellen. Das Thema Usability ist aber ein wichtiger Aspekt für eine Bewertung und soll daher mit in den Katalog aufgenommen werden.

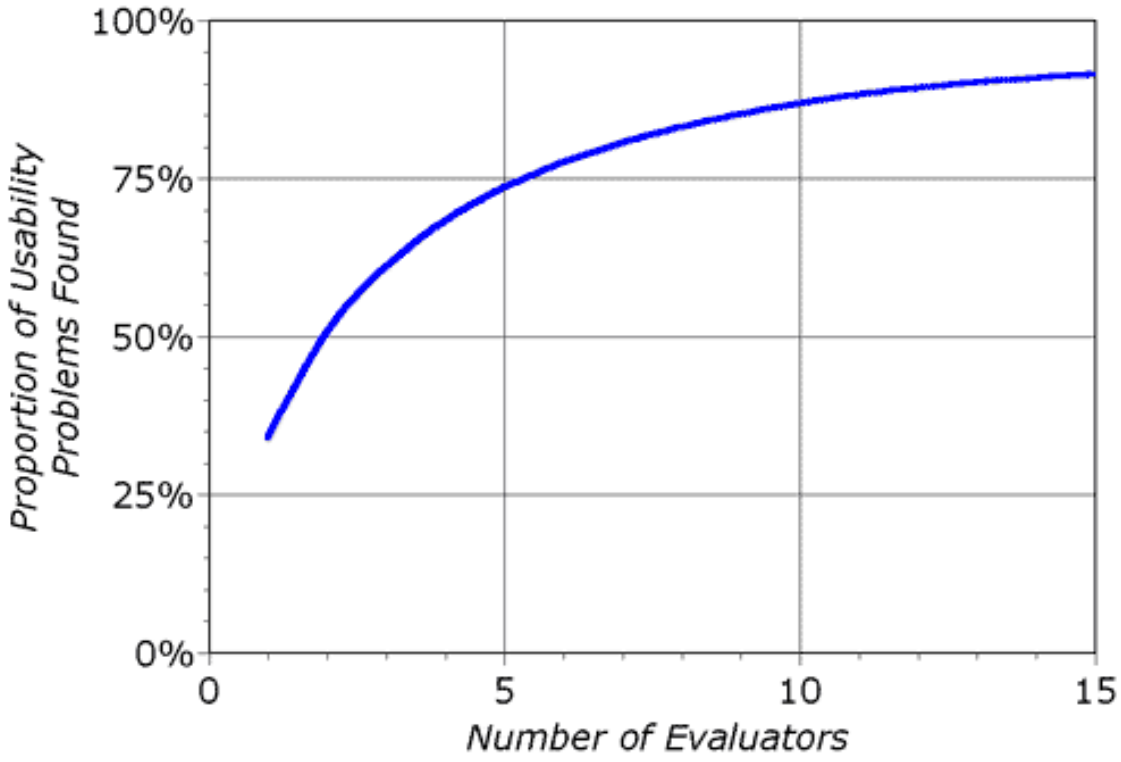


Abbildung 3.1: Verhältnis der gefundenen Probleme zu der Anzahl der Evaluatoren (aus [Nie93])

Die heuristische Evaluation hat nach HOLZINGER (vgl. [Hol05]) einige Vorteile. So kann diese Methode in allen Phasen der Softwareentwicklung eingesetzt werden und die benötigte Zeit für die Evaluation ist im Vergleich zu anderen Varianten der Evaluation eher kurz, auch weil kein zusätzliches Material benötigt wird. Außerdem werden keine Fachnutzer der Software benötigt, da die Heuristiken genereller Art sind. Ein weiteres Argument für die Anwendung der Heuristiken ist, dass diese bereits im Vorfeld von einem Experten für Usability, nämlich von Nielsen selbst, zusammengestellt worden sind und auch allgemein anerkannt werden. Zusammenfassend bietet diese Art der Evaluation eine einfache und schnelle Möglichkeit die Gebrauchstauglichkeit einer Software zu bewerten.

Bei jedem der zehn Heuristiken wird zunächst bei der Bewertung von keinem Usabili-

tyfehler ausgegangen und der Kandidat erhält 4 Punkte (*voll erfüllt*). Für jeden gefundenen Fehler im Sinne der Heuristik wird ein Punkt abgezogen.

Die Einhaltung der einzelnen Heuristiken in Form von Kriterien wird überprüft, indem die Werkzeuge hintereinander mit einer Auswahl typischer Szenarien benutzt werden. Gefundene Fehler werden dem jeweiligen Kriterium zugeordnet und kommentiert. Weil dieses Vorgehen für alle Kriterien identisch ist, wird nicht für jedes einzelne Kriterium ein Indikator definiert, sondern auf diesen Abschnitt verwiesen. Die Anforderungen der Kriterien für eine Bewertung sind jeweils aus der dazugehörigen Beschreibung entnehmbar. Die typischen Szenarien sind:

- Werkzeug aufrufen und sich einen Überblick verschaffen.
- Ein Benutzerkonto erstellen.
- Eine Story mit zwei Tasks innerhalb einer neu angelegten Iteration erstellen.
- Eine Story kommentieren und bearbeiten.
- Nach einem Task suchen.
- Während des Erstellens einer Story die Hilfefunktion aufrufen.

Sollten andere Operationen während der Benutzung ebenfalls Heuristiken verletzten, sind diese ebenfalls mit zu berücksichtigen.

# 4 Kriterienkatalog

Wie im vorherigen Kapitel 3.1 zum Vorgehen beschrieben, basiert die Struktur der Kriterien des Kriterienkatalogs auf einer Beschreibung, einem Einfluss auf die Qualität der Software, einem Indikator anhand dessen das Kriterium beobachtet werden kann und einem Bewertungsschema, welches die Bewertung, bzw. die erreichte Punkteanzahl ergibt.

## 4.1 Fachliche Kriterien

Wie bereits besprochen, sollen die Kriterien aus dem Entwicklungsprozess in zwei Schritten extrahiert werden. Der erste Schritt war die Aufnahme der Szenarien, welche vor Ort zusammen mit den Mitarbeitern erstellt wurden. Die aufbereitete Version dieser Szenarien werde ich im Folgenden darstellen. Es werden Fachbegriffe aus dem Bereich der agilen Softwareentwicklung genutzt, die an dieser Stelle nicht wiederholt erklärt werden.[1] Der zweite darauf aufbauende Schritt, stellt das Extrahieren von geeigneten, auf den Szenarien basierenden, Kriterien dar. Sich wiederholende Umstände in den Szenarien werden nur einmalig in ein Kriterium umgewandelt. Die einzelnen im Entwicklungsprozess involvierten Rollen werden jeweils getrennt in einem Unterabschnitt behandelt. Das Thema Testmanagement bekommt einen eigenen Abschnitt. Das Ergebnis für jeden dieser Abschnitte ist ein vollständiger Satz von Kriterien.

### 4.1.1 Entwickler

Aufgrund des Umfangs der Szenarien der Entwicklerrolle wird im Folgenden in mehreren Teilen alternierend der Entwicklungsprozess beschreiben und die Kriterien extrahiert.

Der Entwicklungsprozess im Fallbeispiel unterteilt sich zeitlich in mehrere Abschnitte. Nach der ersten Anforderungsanalyse ergeben sich in Zusammenarbeit mit dem Kunden Stories, welche den Status *"Definiert"* erhalten. Die Stories werden anschließend geschätzt. Geschätzt wird entweder auf Story- oder auf Taskebene, im Team von mehreren Personen beim sogenannten Schätzpoker. Es werden keine absoluten Zeitangaben geschätzt, sondern relative Aufwandspunkte. Sollte auf Taskebene geschätzt werden, so werden die Stories vorher beim ''Taskifizieren" von zwei Personen in einzelne Tasks heruntergebrochen. Die Schätzung von Task erfolgt in absoluten Zeiteinheiten. Bei der späteren Implementierung wird stets eine der beim Schätzen beteiligten Personen mitarbeiten. Ergeben sich zu einem beliebigem Zeitpunkt Rückfragen zu einer Story, wird diese gekennzeichnet und die entsprechende Frage formuliert. Die Klärung läuft zunächst über den Berater, da dieser durch den direkten Kundenkontakt einen besseren Überblick über

[1] Hier sei auf das Kapitel 2 zur Terminologie verwiesen.

die Spezifikation hat. Sollte der Berater die Frage nicht beantworten können, wendet er sich an den Kunden. Die Klärung der Frage wird in der Story mit dokumentiert.

Es werden nicht unbedingt alle Stories geschätzt, da der Aufwand dafür zu hoch wäre. Stattdessen wird gemeinsam mit dem Kunden priorisiert.

Das Ziel ist eine Menge von geschätzten Stories, die sich zunächst in einem Pool befinden.

**Extrahierte Kriterien:**

| **Kriterium** | Unterstützung des Story-Konzepts |
|---|---|
| **Beschreibung** | Dieses Kriterium bewertet das Werkzeug dahingehend, ob das Konzept der Story unterstützt wird. Dazu zählt eine Repräsentation der Story mit ihren grundlegenden Attributen *Name*, *Beschreibung der Funktionalität* und einer *Schätzung*. Dazu kommen die Attribute *Status*, *Priorität* und eventuelle *Beobachter*. |
| **Motivation** | Ist das Konzept nicht Teil des Werkzeugs, erfüllt es eines der grundlegenden Konzepte der agilen Softwareentwicklung nicht. Das Vorhandensein ist ein Muss. |
| **Indikator** | Das Werkzeug ist auf dieses Konzept dahingehend zu überprüfen, ob es das Erstellen von Stories zulässt. Je mehr in der Beschreibung genannte Funktionen vorhanden sind, desto besser wird das Konzept unterstützt. Der Optimalfall gestaltet sich in Form von Stories, die in ihrer Ausgestaltung voll konfigurierbar sind, um auf sich ändernde Prozesse eingehen zu können. |
| **Bewertung** | • *voll erfüllt:* Stories sind voll konfigurierbar.<br>• *überwiegend erfüllt:* Stories enthalten alle in der Beschreibung genannten Attribute.<br>• *teilweise erfüllt:* Stories enthalten lediglich die in der Beschreibung genannten grundlegenden Attribute.<br>• *nicht erfüllt:* Stories werden nicht unterstützt. |

| **Kriterium** | Schätzen in relativen Aufwandspunkten und absoluten Zeiteinheiten |
|---|---|
| **Beschreibung** | Dieses Kriterium stellt sicher, dass Stories anhand von Aufwandspunkten und Tasks anhand von Zeitangaben geschätzt werden können. |
| **Motivation** | Das Schätzen gehört zu den Stories als grundlegende agile Methode mit dazu. Andere agile Methoden basieren auf einer erfolgten Schätzung. |
| **Indikator** | Stories beinhalten ein Attribut für die Schätzung in Aufwandspunkten und Tasks Attribute für die Schätzung in Zeiteinheiten. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug überlässt dem Benutzer die Modellierung der Zeitschätzung. Sowohl absolute, als auch relative Schätzungen werden so ermöglicht.<br>• *überwiegend erfüllt:* Das Werkzeug verhält sich wie in der Beschreibung.<br>• *teilweise erfüllt:* Es lassen sich lediglich Stories, oder lediglich Tasks schätzen.<br>• *nicht erfüllt:* Eine Angabe eine Schätzung wird durch das Werkzeug nicht unterstützt. |

| **Kriterium** | Stories & Tasks haben Zustände |
|---|---|
| **Beschreibung** | Der Zustand einer Story oder eines Tasks drückt die aktuelle Position im Entwicklungsprozess aus. |
| **Motivation** | Der Übersicht halber möchte man Stories und Tasks einem Status zuordnen, um einen Überblick zu behalten. Es ist beispielsweise schnell ersichtlich, welche Stories noch geschätzt werden müssen, oder welche bereits implementiert werden können. Nach Möglichkeit soll es frei einstellbare Zustände geben. |
| **Indikator** | Stories und Tasks haben ein Attribut Status, der sie unterschiedlichen Kategorien zuordnet. Gewünschte Kategorien sind *Entwurf*, *Definiert*, *Geschätzt*, *Geplant*, *Implementiert*, *Verifiziert* und *Akzeptiert*. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Das Zustandsattribut einer Story lässt sich durch das Werkzeug frei konfigurieren. Es können beliebige Zustände konfiguriert werden.<br>• *überwiegend erfüllt:* Alle im Indikator genannten Felder sind vorhanden.<br>• *teilweise erfüllt:* Weniger als $\frac{3}{4}$ der genannten Zustände sind wählbar.<br>• *nicht erfüllt:* Stories haben keine Zustände. |

| **Kriterium** | Zuordbarkeit von Personen zu Stories |
|---|---|
| **Beschreibung** | Stories haben ein Attribut welches eine Person angibt. |
| **Motivation** | Da Personen, die am Schätzen einer Story beteiligt waren, später auch mit implementieren sollen, müssen sich Stories Personen zuordnen lassen. Prinzipiell werden die beiden Konzepte des Implementierenden und des Beobachters genutzt, wobei jede Position auch durch mehrere Personen besetzt werden kann. |
| **Indikator** | Stories und Tasks verfügen über diese Möglichkeit, wie in der Motivation beschrieben. |
| **Bewertung** | • *voll erfüllt:* Voll konfigurierbar.<br>• *überwiegend erfüllt:* Sowohl für Stories, als auch für Tasks, werden Attribute für den Implementierenden und für Beobachter zur Verfügung gestellt. Eventuell sind die Attribute anders benannt.<br>• *teilweise erfüllt:* Nicht alle Möglichkeiten für Zuordnungen, wie in der Motivation beschrieben, stehen zur Verfügung.<br>• *nicht erfüllt:* Es gibt keine Möglichkeit einer Story oder einem Tasks Personen zuzuordnen. |

| | |
|---|---|
| **Kriterium** | Die Möglichkeit Tasks als hierarchische Aufgabenteilung für Stories zu verwenden |
| **Beschreibung** | Für Stories können jeweils einzelne Tasks erstellt werden, die der Story zugehörig sind und somit eine Hierarchie bilden. |
| **Motivation** | Eine Story ist oft zu abstrakt oder zu komplex, als dass sie vernünftig geschätzt und implementiert werden kann. Daher soll es die Möglichkeit geben eine Story im Sinne einer Aufgabenteilung in einzelne Tasks herunterzubrechen. Durch den geringeren Umfang und einer detaillierteren Beschreibung des Sachverhaltes von Tasks wird sowohl eine Schätzung, als auch die Implementierung vereinfacht. |
| **Indikator** | Zu Stories lassen sich Tasks erstellen. |
| **Bewertung** | • *voll erfüllt:* Tasks lassen sich erstellen und Stories zuordnen. Sie lassen sich in ihren Attributen voll konfigurieren.<br>• *überwiegend erfüllt:* Tasks lassen sich erstellen und Stories zuordnen. Tasks haben aber eine im Vorfeld definierte und nicht änderbare Struktur.<br>• *teilweise erfüllt:* Tasks existieren zwar, doch lassen sie sich Stories nicht zuordnen.<br>• *nicht erfüllt:* Das Konzept von Tasks wird vom Werkzeug nicht unterstützt. |

| | |
|---|---|
| **Kriterium** | Kommentierbarkeit von Stories |
| **Beschreibung** | Stories sollen freie Textfelder für das Hinzufügen von Kommentaren erhalten. |
| **Motivation** | Falls Anmerkungen oder Rückfragen zu einer Story aufkommen, sollen diese dokumentiert gestellt und beantwortet werden. Im Sinne des *Collective Code Ownerships* und des *Truckfactors* soll das Wissen dem gesamten Team nachvollziehbar zur Verfügung gestellt werden. |

| | |
|---|---|
| **Indikator** | Stories erlauben das Formulieren von Kommentaren. Idealerweise findet dies strukturiert statt, sodass eine Diskussion nachvollzogen werden kann. Zusätzlich wäre es optimal, wenn Tasks diese Funktion ebenfalls unterstützen. Außerdem soll es möglich sein Dateien anzuhängen. |
| **Bewertung** | • *voll erfüllt:* Freie Konfigurierbarkeit von freien Textfeldern für Stories und Tasks mit einer strukturierten Diskussionsansicht, sowie die Unterstützung für Dateianhänge.<br>• *überwiegend erfüllt:* Die Möglichkeit einer Diskussion werden für Stories und Tasks, inklusive Dateianhänge, angeboten.<br>• *teilweise erfüllt:* Diskussion werden entweder nur für Stories oder nur für Tasks angeboten. Dateianhänge werden nicht unterstützt.<br>• *nicht erfüllt:* Eine Funktion für Diskussionen wird nicht angeboten. |

| **Kriterium** | Priorisierbarkeit von Stories |
|---|---|
| **Beschreibung** | Stories sollen vor einer Schätzung priorisiert werden können. |
| **Motivation** | Nach der Anforderungsanalyse sind alle Stories ausformuliert. Sollten die Anzahl der Stories zu hoch sein, wird nur ein Teil geschätzt. Dazu müssen die Stories für das Schätzen priorisiert werden. |
| **Indikator** | Stories haben ein Attribut für die Priorisierung. Anhand dessen kann auch gefiltert und sortiert werden. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Priorisierungen werden voll unterstützt und lassen sich über Filter und Sortierungen entsprechend anzeigen. Der Grad der Priorisierungen ist voll konfigurierbar.<br>• *überwiegend erfüllt:* Priorisierungen werden voll unterstützt und lassen sich über Filter und Sortierungen entsprechend anzeigen. Die möglichen Priorisierungen sind aber fest definiert.<br>• *teilweise erfüllt:* Priorisierungen werden unterstützt, aber es gibt Mängel diese zu filtern und zu sortieren.<br>• *nicht erfüllt:* Das Werkzeug bietet keine Möglichkeit Stories zu priorisieren. |

| **Kriterium** | Ablageplätze für Stories, unabhängig vom Status |
|---|---|
| **Beschreibung** | Es soll möglich sein, Stories mit ihren Tasks in einem beliebigen "Pool" zu lagern. |
| **Motivation** | Es gibt beispielsweise Stories die keiner Iteration angehören. Diese will man an einer geeigneten Stelle auslagern. |
| **Indikator** | Das Werkzeug erlaubt das freie Erstellen von Orten für Stories und unterstützt das Zuordnen von Stories zu diesen Orten. |
| **Bewertung** | • *voll erfüllt:* Es lassen sich beliebige Orte anlegen. Stories lassen sich in diese verlegen.<br>• *überwiegend erfüllt:* Alle Stories die keiner Iteration zugeordnet sind lagern in einem "Nicht zugeordnet" Pool.<br>• *teilweise erfüllt:* Das Fehlen von Orten außerhalb von Iterationen wird damit umgangen, eine Iteration für diese Sorte von Stories anzulegen.<br>• *nicht erfüllt:* Stories kann es nicht unabhängig von Iterationen geben. |

Die geschätzten Stories werden nun vom Kunden, mit Unterstützung des Beraters, zu Iterationen hinzugefügt. Dabei darf die maximale Anzahl der möglichen Aufwandspunkte nicht überschritten werden. Sollten nicht alle Aufwandspunkte aufgebraucht worden sein, werden *"Slackstories"* definiert, die am Ende einer Iteration ausgeführt werden. Die Stories einer Iteration werden von den Entwicklern in eine Reihenfolge gebracht. Außerdem machen die Entwickler auf etwaige Abhängigkeiten zwischen den Stories aufmerksam. Das Ergebnis ist eine definierte Iteration.

**Extrahierte Kriterien:**

| **Kriterium** | Definierbakeit von Iterationen |
|---|---|
| **Beschreibung** | Das Werkzeug erlaubt das Erstellen von Iterationen mit Start- und Enddatum sowie der zur Verfügung stehenden Aufwandspunkte. Nach Möglichkeit soll es eine frei definierbare Anzahl von Attributen geben. |
| **Motivation** | Der Entwicklungsprozess sieht vor, die Entwicklung auf Iterationen aufzuteilen. Dies erlaubt eine verbesserte Schätzung für den gesamten Entwicklungsprozess. |
| **Indikator** | Das Werkzeug bietet die Funktion Iterationen zu verwalten. Dazu gehört diese zu erstellen, starten, stoppen, beenden und editieren. Es soll für Iterationen Attribute geben, die Start- und Enddaten angeben. Außerdem soll die Anzahl der verbrauchten-, sowie die, der zur Verfügung stehenden Aufwandspunkte, dargestellt werden. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug bietet alle im Indikator genannten Funktionen, inklusive der freien Definition der Attribute.<br>• *überwiegend erfüllt:* Das Werkzeug bietet alle im Indikator genannten Funktionen.<br>• *teilweise erfüllt:* Iterationen werden starr unterstützt. Attribute wie Daten oder Aufwandspunkte fehlen.<br>• *nicht erfüllt:* Iterationen werden nicht unterstützt. |

| **Kriterium** | Zuordbarkeit von Stories zu Iterationen |
|---|---|
| **Beschreibung** | Stories lassen sich einer Iteration zuordnen. |
| **Motivation** | Iterationen zeichnen sich durch die enthaltenen Stories aus. Das Werkzeug sollte also die Funktion bieten, Stories mindestens einer Iteration zuzuordnen. Sollte eine Story nicht während einer Iteration abgearbeitet worden sein, so muss diese in einer andere Iteration verschiebbar sein. |
| **Indikator** | Stories lassen sich einer Iteration zuordnen. Bei Bedarf kann diese Zuordnung auch geändert werden. |
| **Bewertung** | • *voll erfüllt:* Stories lassen sich beliebigen Iterationen zuordnen.<br>• *überwiegend erfüllt:* Das Hinzufügen von Stories zu Iterationen ist auf irgendeine Art und Weise beschränkt.<br>• *teilweise erfüllt: entfällt*<br>• *nicht erfüllt:* Stories lassen sich keinen Iterationen zuordnen. |

| **Kriterium** | Kennzeichenbarkeit von Slackstories |
|---|---|
| **Beschreibung** | Slackstories müssen irgendwie gekennzeichnet werden, um sie als solche zu erkennen. Neben Slackstories könnte es noch andere Arten von Stories geben, daher ist eine Art freie Typisierung von Stories wünschenswert. |
| **Motivation** | Slackstories sind solche Stories, die zum Ende einer Iteration bearbeitet werden, sollten noch Aufwandspunkte übrig sein. Damit diese auch wirklich nur unter diesen Umständen bearbeitet werden ist eine Kennzeichnung wichtig. |
| **Indikator** | Slackstories können als solche gekennzeichnet werden. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Alle in der Beschreibung genannten Punkte werden durch die Anwendung unterstützt.<br>• *überwiegend erfüllt:* Slackstories werden unterstützt, doch fehlt eine freie Typisierung von Stories.<br>• *teilweise erfüllt:* Eine Kennzeichnung wird nur schlecht unterstützt. Eventuell geschieht dies durch einen speziellen Status.<br>• *nicht erfüllt:* Eine spezielle Kennzeichnung von Stories wird vom Werkzeug nicht angeboten. |

| **Kriterium** | Definierbarkeit der Reihenfolge von Stories einer Iteration |
|---|---|
| **Beschreibung** | Stories einer Iteration werden mit einer Reihenfolge versehen in der sie bearbeitet werden sollen. Die Reihenfolge soll entsprechend visualisiert werden. |
| **Motivation** | Ist eine Reihenfolge gewünscht, so soll diese durch das Werkzeug auch unterstützt und kenntlich gemacht werden. Beispielsweise werden Reihenfolgen benötigt, wenn ein Entwickler sich eine Liste der zu erledigenden Dinge aufstellt. |
| **Indikator** | Das Werkzeug muss das Konzept einer Reihenfolge unterstützen. |
| **Bewertung** | • *voll erfüllt:* Stories einer Iteration lassen sich einer Reihenfolge zuordnen. Eine Story kann Bestandteil von beliebig vielen Reihenfolgen sein.<br>• *überwiegend erfüllt:* Stories einer Iteration lassen sich einer Reihenfolge zuordnen.<br>• *teilweise erfüllt:* Es wird sich damit beholfen den Stories eine Priorisierung zu geben.<br>• *nicht erfüllt:* Reihenfolgen von Stories werden vom Werkzeug nicht unterstützt. |

| **Kriterium** | Definierbarkeit von Abhängigkeiten der Stories einer Iteration |
|---|---|
| **Beschreibung** | Abhängigkeiten zwischen Stories sollen vom Werkzeug entsprechend visualisiert werden. |
| **Motivation** | Abhängigkeiten zwischen Stories sind ganz natürlich. So muss beispielsweise zunächst die Story bearbeitet werden, die das Einrichten der Datenbank spezifiziert, bevor eine Story zur Kommunikation mit der Datenbank bearbeitet werden kann. |
| **Indikator** | Das Werkzeug unterstützt das Prinzip Abhängigkeiten zwischen Stories zu definieren, was auch entsprechend in jeder Story angegeben wird. |
| **Bewertung** | • *voll erfüllt:* Stories einer Iteration können in Abhängigkeit zueinander stehen. Eine Story kann Bestandteil von beliebig vielen Abhängigkeiten sein. Wird eine Story bearbeitet, die den Abschluss einer anderen Story voraussetzt, wird darauf hingewiesen.<br>• *überwiegend erfüllt:* Stories einer Iteration können in Abhängigkeit zueinander stehen. Es gibt aber Mängel die Abhängigkeiten dem Benutzer kenntlich zu machen.<br>• *teilweise erfüllt: entfällt*<br>• *nicht erfüllt:* Abhängigkeiten werden nicht unterstützt. |

Ist die Iteration definiert, beginnt ihre Bearbeitung. Die Stories werden anhand der definierten Reihenfolgen und Abhängigkeiten abgearbeitet. Im Zuge dessen wechselt der Status auf *"In Bearbeitung"*. Sind einzelne Tasks spezifiziert worden, so findet die Bearbeitung entlang dieser statt. Während der Bearbeitung der Stories halten die Entwickler die verwendete Zeit für Stories und Tasks fest. Dies gilt auch für das Programmieren im Paar. Der Fortschritt von Stories soll visualisiert werden, weil dadurch der Fortschritt der gesamten Iteration ermöglicht wird. Es soll gewarnt werden, falls die aktuell verbrauchten Aufwandspunkte über der Schätzung liegen. Eine Story ist abgeschlossen und wechselt auf den Status *"Implementiert"*, wenn folgende Teilaspekte fertiggestellt wurden: Implementierung, Schreiben von Tests, erfolgreiches Durchlaufen der Tests, Tests in der Anwendung, keinerlei Compilerwarnungen, Dokumentation in Form von Quelltextkommentaren und bei Bedarf von UML-Diagrammen, SQL-Skripten und Wiki-Einträgen. Anschließend wird die Story gemeinsam mit einer anderen Person verifiziert und erhält bei

einer erfolgreichen Verifizierung den Status *"Verifiziert"*. Sollte die in der Story beschriebene Funktionalität auch beim Kunden erfolgreich laufen, erhält die Story den Status *"Kundenfertig"*. Nur Stories mit diesem Status dürfen für ein Release verwendet werden.

Im Laufe einer Iterationen werden auch Pausentage für Spikes und andere technische Dinge eingelegt. Diese laufen zwar zeitgleich zu einer Iteration, gehören dieser aber nicht an.

**Extrahierte Kriterien:**

| **Kriterium** | Dokumentierbarkeit der verwendeten Zeit |
|---|---|
| **Beschreibung** | Stories werden mit Attributen versehen, welche die für sie verwendete Zeit wiedergeben. |
| **Motivation** | Das Schätzen von Aufwänden kann nur verbessert werden, wenn die im Vorfeld geschätzte Zeit mit der am Ende wirklich verwendeten Zeit verglichen wird. |
| **Indikator** | Stories, sowie Tasks erlauben das Angeben von Arbeitszeiten. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug unterstützt ein ausführliches Zeitmanagement. Auch für das Programmieren im Paar. Darüber hinaus werden die verschiedenen Zeiten und Schätzungen entsprechend visualisiert.<br>• *überwiegend erfüllt:* Stories und Tasks erlauben das Eintragen von Zeiten. Auch für das Programmieren im Paar.<br>• *teilweise erfüllt:* Nur Stories oder Tasks erlauben das Eintragen von Zeiten.<br>• *nicht erfüllt:* Das Festhalten von Arbeitszeiten für Stories wird vom Werkzeug nicht unterstützt. |

| **Kriterium** | Visualisierbarkeit des Fortschritts einer Iteration |
|---|---|
| **Beschreibung** | Der Fortschritt einer Iteration wird geeignet visualisiert. Dazu gehören auch Warnungen, falls Schätzungen fehlerhaft waren. |

| | |
|---|---|
| **Motivation** | Eine Visualisierung ermöglicht eine schnelle Methode den Fortschritt einer Iteration zu erkennen. Beispielsweise lässt sich anhand eines *"Burn-Down-Charts"* erkennen, ob die Geschwindigkeit des Voranschreitens im Soll liegt, oder abweicht. |
| **Indikator** | Der Fortschritt einer Iteration wird auf eine geeignete Art visualisiert. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug bietet voll konfigurierbare Visualisierungen<br>• *überwiegend erfüllt:* Einige fest vorgegebene Visualisierungen werden geboten.<br>• *teilweise erfüllt:* Der Fortschritt wird nur über einfache Grafiken visualisiert, die nur zum Teil ausreichend sind.<br>• *nicht erfüllt:* Eine Visualisierungen des Fortschritts gibt es nicht. |

| **Kriterium** | Definierbarkeit von speziellen Typen von Stories |
|---|---|
| **Beschreibung** | Das Werkzeug soll ermöglichen spezielle Typen von Stories zu definieren, die abseits von Iterationen existieren. |
| **Motivation** | Spikes und technische Dinge können ebenso als Story definiert werden, wie normale Stories. Der Unterschied ist aber, dass normale Stories gemeinsam mit dem Kunden verfasst werden und dieser für die Implementation innerhalb einer Iteration auch entsprechend bezahlt. Spikes und technische Dinge werden aber vom Entwicklungs- und Beraterteam selber definiert. Sie sind nicht Teil des Kundenauftrags. |
| **Indikator** | Das Werkzeug bietet das Definieren spezieller Stories an, die abseits von Iterationen existieren können. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Die in der Beschreibung vorgestellten Eigenschaften werden vom Werkzeug voll unterstützt. Darüber hinaus lassen sich beliebige Typen definieren.<br>• *überwiegend erfüllt:* Die in der Beschreibung vorgestellten Eigenschaften werden vom Werkzeug voll unterstützt.<br>• *teilweise erfüllt:* Es wird eine vordefinierte Kategorie für solche Stories angeboten.<br>• *nicht erfüllt:* Spezielle Typen von Stories werden nicht unterstützt. |

Ist eine Iteration terminlich abgeschlossen, obwohl nicht alle Stories aufgrund von fehlerhaften Schätzungen bearbeitet werden konnten, wandern diese zurück in den Pool aller unbearbeiteten Stories. Es gilt mit dem Kunden abzustimmen, ob diese Stories, neben den anderen unbearbeiteten Stories, mit in die nächste Iteration aufgenommen werden sollen. Stories die nur zum Teil abgeschlossen wurden, werden gesplittet und in die nächste Iteration übernommen. Zusätzlich wird anhand der Statistik über die geschätzten und die tatsächlich verbrauchten Aufwandspunkten überprüft, ob die aktuelle Schätzung der anderen Stories noch realistisch ist.

**Extrahierte Kriterien:**

| **Kriterium** | Der Ort einer Story kann beliebig wechseln |
|---|---|
| **Beschreibung** | Eine Story kann einem beliebigen Ort angehören. Beispiele sind Iterationen oder Pools außerhalb von Iterationen. Zu jedem Zeitpunkt kann der Ort verändert werden |
| **Motivation** | Wird eine Story innerhalb einer Iteration nicht abgeschlossen, so kann diese der nächsten Iteration zugeordnet werden, oder zurück in den Pool aller nicht-zugeordneten Stories wandern. |
| **Indikator** | Das Werkzeug muss, wie in der Beschreibung dargestellt, das Ändern von Orten für Stories zulassen. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Das Verschieben wird komplett unterstützt.<br>• *überwiegend erfüllt:* Wenige Ausnahmen sind beim Verschieben zu beachten.<br>• *teilweise erfüllt:* Es gibt Einschränkungen beim Verschieben.<br>• *nicht erfüllt:* Das Verschieben wird vom Werkzeug nicht unterstützt. |

| **Kriterium** | Aufteilbarkeit von Stories |
|---|---|
| **Beschreibung** | Stories lassen sich in ihrem Inhalt auf zwei Stories aufteilen. |
| **Motivation** | Wurde eine Story zum Ende einer Iteration nicht komplett abgeschlossen, soll der bereits abgeschlossene Teil in der beendeten Iteration verbleiben, während der noch offene Teil durch ein aufsplitten der Story in die nächste Iteration übergeht. |
| **Indikator** | Das Werkzeug bietet die Funktion der Aufteilbarkeit von Stories. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug verhält sich genau so, wie in der Beschreibung dargelegt.<br>• *überwiegend erfüllt:* Es gibt einige Besonderheiten, die beim Aufteilen zu beachten sind.<br>• *teilweise erfüllt:* Das Aufteilen von Stories wird durch das Spezifizieren einer neuen Story, sowie dem Verschieben von noch offenen Tasks ermöglicht.<br>• *nicht erfüllt:* Eine Aufteilung von Stories ist nicht möglich. |

| **Kriterium** | Reflektierbarkeit über Aufwandspunkte |
|---|---|
| **Beschreibung** | Anhand einer Statistik kann überprüft werden, ob die Schätzung einer Iteration mit den wirklich geleisteten Aufwandspunkten übereinstimmt. |

| | |
|---|---|
| **Motivation** | Durch die Statistik kann eingeschätzt werden, ob die aktuelle Schätzung von anderen Stories noch realistisch ist. |
| **Indikator** | Das Werkzeug bietet die Möglichkeit anhand von Statistiken über geschätzte und geleistete Aufwandspunkte zu reflektieren. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug bietet eine umfassende Unterstützung, um den Vorgang des Reflektierens zu vereinfachen.<br>• *überwiegend erfüllt:* Das Werkzeug bietet einige simple Mittel, die das Reflektieren ermöglichen.<br>• *teilweise erfüllt:* Das Werkzeug bietet lediglich beschränkte Mittel für die Reflexion.<br>• *nicht erfüllt:* Es werden von Werkzeug kein Mittel der Unterstützung für die Reflexion angeboten. |

### 4.1.2 Berater

Die Arbeit des Beraters beginnt bereits vor der ersten Iteration. Aus Interviews mit dem Kunden, sowie einer Analyse des Altsystems[2] hinsichtlich der Applikation und der Datenbank, werden Usecases erstellt. Anhand der Usecases werden erste GUI-Entwürfe erstellt und mit dem Kunden besprochen. Für Details in der zu erarbeitenden Spezifikation werden Gespräche mit den jeweiligen Fachabteilungen geführt. Aus den Usecases entstehen Anforderungen, welche dokumentiert werden. Wie aus Sicht des Entwicklers bereits beschrieben, entstehen aus den Anforderungen die Stories.

Die Kommunikation mit dem Kunden vor Ort findet anfangs zwei Mal wöchentlich für je 4h statt, später je nach Bedarf. Andere Hilfsmittel für die Kommunikation und Dokumentation sind im aktuellen Projekt des Fallbeispiels ein Kundenportal, in dem das Handbuch, ein Glossar, die Usecases, Informationen zu den Altsystemen und Protokolle zu Besprechungen aufbewahrt werden.

Obwohl die Kommunikation mit dem Kunden durch Hilfsmittel wie ein Wiki und ein Forum unterstützt werden, schlägt diese Art der Kommunikation oft fehl. Es wird oft auf die direkte Kommunikation, entweder vor Ort, oder per Telefon zurückgegriffen.

Der Berater bildet die Schnittstelle zwischen den Entwicklern und den Kunden. Daher muss dieser einen guten Überblick über den Fortschritt des Projekts besitzen und dem Kunden auch Rechenschaft ablegen können. Er ist ebenso Ansprechpartner für Probleme

[2] Wie in der Einleitung erwähnt, hat sich das Unternehmen im Fallbeispiel darauf spezialisiert, Altsysteme durch moderne Lösungen abzulösen.

mit der zu entwickelnden Anwendung. Dazu zählen Fehler in der Spezifikation sowie Bugs.

**Extrahierte Kriterien:**

| **Kriterium** | Dokumentierfähigkeit |
|---|---|
| **Beschreibung** | Das Kriterium beschreibt die Anforderung an ein Werkzeug, das Projekt ausreichend dokumentieren zu können. Dazu gehören das Handbuch, ein Glossar, die Usecases, Informationen zu den Altsystemen und Protokolle zu Besprechungen. Diese Artefakte müssen sich im Werkzeug geeignet einbinden lassen. |
| **Motivation** | Als Vorteil bei der im Werkzeug integrierten Aufbewahrung ergibt sich eine einfache Referenzierung auf Inhalte. Dies gilt für beide Richtungen. Beispielsweise können Stories auf Usecases referenzieren, und Usecases auf Stories. |
| **Indikator** | Das Werkzeug bietet eine Form der Dokumentation. |
| **Bewertung** | • *voll erfüllt:* Möglichkeiten zur Dokumentation sind umfangreich gegeben. Es gibt Beispielsweise ein Wiki, ein Forum und ein Archiv für Dokumente und Dateien.<br>• *überwiegend erfüllt:* Möglichkeiten zur Dokumentation sind zwar vorhanden, aber weniger umfangreich.<br>• *teilweise erfüllt:* Möglichkeiten zur Dokumentation sind kaum vorhanden.<br>• *nicht erfüllt:* Möglichkeiten zur Dokumentation werden vom Werkzeug nicht angeboten. |

| **Kriterium** | Kommunikation mit dem Kunden |
|---|---|
| **Beschreibung** | Es soll die Möglichkeit bestehen über beliebige Themen über das Werkzeug mit dem Kunden zu kommunizieren. |
| **Motivation** | Vorteile hier ist wieder die Möglichkeit Stories zu referenzieren, beziehungsweise auf die Dokumentation zu verweisen. |

| | |
|---|---|
| **Indikator** | Es werden Kommunikationstools wie Kommentare, persönliche Nachrichten, ein Forum oder ein Chat angeboten. |
| **Bewertung** | • *voll erfüllt:* Sämtliche im Indikator aufgezählte Tools sind vorhanden.<br>• *überwiegend erfüllt:* Ein Tool fehlt im Werkzeug.<br>• *teilweise erfüllt:* Wenigstens ein Tool wird angeboten.<br>• *nicht erfüllt:* Eine Möglichkeit der Kommunikation gibt es nicht. |

| | |
|---|---|
| **Kriterium** | Visualisierbarkeit des gesamten Projektfortschritts |
| **Beschreibung** | Der Projektstand wird anhand von Grafiken visualsiert. Dabei sind verschiedenste Varianten denkbar. |
| **Motivation** | Der Kunde hat ein Interesse sich über den Fortschritt des Projekts zu informieren. |
| **Indikator** | Das Werkzeug bietet Visualisierungen verschiedener Daten an. |
| **Bewertung** | • *voll erfüllt:* Es lassen sich sämtliche, vorhandene Daten in einer konfigurierbaren Visualisierung anzeigen.<br>• *überwiegend erfüllt:* Die wichtigsten Visualisierung wie *"Burn-Down-Charts"* und das Verhältnis der Anzahl der abgeschlossenen Stories zu der Gesamtzahl der Stories sind enthalten.<br>• *teilweise erfüllt:* Nur wenige Visualisierungen sind enthalten.<br>• *nicht erfüllt:* Visualisierungen werden vom Werkzeug nicht unterstützt. |

| **Kriterium** | Bugtracking |
|---|---|
| **Beschreibung** | Das Werkzeug bietet eine dedizierte Funktion für das Verwalten von Bugs. |
| **Motivation** | Bugs werden oft unabhängig von Stories und Iterationen entdeckt. Das Werkzeug soll ermöglichen mit diesem Aspekt entsprechend umzugehen. Aus Bugs entstehen teilweise neue Stories, wodurch sich unter anderem die Integration eines Bugtrackers in das Werkzeug begründen lässt. |
| **Indikator** | Das Werkzeug bietet, wie oben beschrieben, eine Möglichkeit mit Bugs umzugehen. Dazu können Aspekte wie das Erfassen, Dokumentieren und Verfolgen von Bugs gehören. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug unterstützt eine Vielzahl der beschriebenen Eigenschaften.<br>• *überwiegend erfüllt:* Das Bugtrackingtool innerhalb des Werkzeugs ist durch einige wenige fehlende Aspekte beschränkt.<br>• *teilweise erfüllt:* Es wird lediglich ein simples und kaum konfigurierbares Bugtracking ermöglicht.<br>• *nicht erfüllt:* Eine dedizierte Bugtrackingfunktion wird nicht angeboten. |

### 4.1.3 Testmanagement

Weil das Thema ”Testen” rollenübergreifend ist, wird es in diesem Abschnitt gesondert behandelt. Das Unternehmen im Fallbeispiel zeichnet sich durch extensives testen aus. Es findet auf fast sämtlichen Ebenen statt. Bereits vor der eigentlichen Entwicklung wird versucht, die zusammen mit dem Kunden ausgearbeiteten Usecases und Spezifikationen auf inhaltliche Korrektheit und etwaige Unstimmigkeiten zu testen.

Während der Entwicklung werden die technischen Spezifikationen einem Review unterzogen. Die Implementierung zeichnet sich durch Unittests für jede Klasse und Story, sowie durch storyübergreifende Integrationstests aus, die sich inhaltlich auf die Spezifikationen und Anforderung stützen. Für während der Entwicklung auftretende Bugs wird ein Bugtracking verwendet. Die Anwesenheit und die Korrektur eines Bugs wird durch einen Test nachgewiesen. Für Usecases werden Tests erstellt, die entweder automatisiert

per Selenium oder manuell ausgeführt werden. Die Verwendung eines Buildservers unterstützt die Entwickler bei Unittests, indem alle vorhandenen Unittests in einem sogenannten 10-Minute-Build massenhaft ausgeführt werden. Außerdem werden auf dem Buildserver Tests gegen Datenbanken und Backends durchgeführt.

Nach Herausgabe eines Releases, wird seitens des Kunden ein User-Acceptance-Test durchgeführt, um zu überprüfen, ob wirklich alle Funktionen fehlerfrei enthalten sind. Außerdem werden im Betrieb Defects auftreten, die wieder entsprechend mit Tests abgesichert werden müssen.

| **Kriterium** | Unterstützung von Tests auf multiplen Ebenen |
|---|---|
| **Beschreibung** | Das Werkzeug soll die Möglichkeit eines kompletten Managements von Tests anbieten. Dazu zählen alle in der Einleitung zu diesem Kapitel erwähnten Punkte. Das Werkzeug soll also umfassende Verwaltungsmöglichkeiten für dieses Thema bieten. |
| **Motivation** | Ein integriertes Testmanagement erhöht die Übersichtlichkeit und Kontrollierbarkeit von Tests der einzelnen Ebenen und gewährleistet einen besseren Gesamtüberblick. Des Weiteren können Tests durch Querreferenzen auf die Dokumentation, wie z.B. Spezifikation und Usecases, besser fundiert werden. |
| **Indikator** | Das Werkzeug soll eine komplette Lösung zum Management von Tests anbieten. |
| **Bewertung** | • *voll erfüllt:* Ein ausführliches Testmanagement für beliebige Ebenen wird integriert vom Werkzeug zur Verfügung gestellt.<br>• *überwiegend erfüllt:* Das Testmanagement beinhaltet die meisten der oben genannten Ebenen.<br>• *teilweise erfüllt:* Nur wenige der oben genannten Funktionen werden vom Werkzeug erfüllt.<br>• *nicht erfüllt:* Das Werkzeug bietet keinerlei Unterstützung zum Thema Testmanagement. |

| **Kriterium** | Templateunterstützung |
|---|---|
| **Beschreibung** | Für das Schreiben von Stories, Tasks oder auch Bugs und Defects wird vom Werkzeug die Verwendung eines Templates als Grundgerüst für wiederkehrende Beschreibungen angeboten. |
| **Motivation** | Das Beschreiben von beispielsweise Bugs und darauf aufbauend deren Tests hält sich an ein genaues Schema. Zum einen möchte man den Benutzer nicht ständig das Schema abtippen lassen, zum anderen möchte man nicht, dass vergessen wird relevante Informationen anzugeben. Daher ist es sinnvoll verschiedene Templates zur Verfügung zu stellen. |
| **Indikator** | Überprüfen, ob das Werkzeug das Konzept von Templates für textuelle Beschreibungen unterstützt. |
| **Bewertung** | • *voll erfüllt:* Es werden beliebig viele Templates auf allen möglichen Ebenen unterstützt. Templates sind anpassbar.<br>• *überwiegend erfüllt:* Anpassbare Templates werden zwar Angeboten, doch sind sie nicht in allen Situationen nutzbar.<br>• *teilweise erfüllt:* Das Werkzeug gibt einige statische und nicht anpassbare Templates vor.<br>• *nicht erfüllt:* Templates werden nicht unterstützt. |

| **Kriterium** | Referenzierbarkeit von Quelltexten |
|---|---|
| **Beschreibung** | Das Werkzeug bietet die Möglichkeit ein Versionsverwaltung für Quelltexte einzubinden. Es kann auf beliebige Stellen im Quelltext referenziert werden. |
| **Motivation** | Werden Bugs beschrieben und besteht bereits der Verdacht, dass eine bestimmte Stelle im Quelltext für den Bug verantwortlich ist, kann direkt darauf referenziert werden. |
| **Indikator** | Das Werkzeug bietet die Möglichkeit Repositories einzubinden und darauf zu referenzieren. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Beliebig viele Repositories können eingebunden und verwaltet werden. Der Quelltext kann direkt begutachtet und referenziert werden.<br>• *überwiegend erfüllt:* Es kann ein Repository eingebunden werden. Der Quelltext kann direkt begutachtet und referenziert werden.<br>• *teilweise erfüllt:* Es kann zwar ein Repository eingebunden werden, doch besteht die Referenziermöglichkeit nur auf Dateiebene.<br>• *nicht erfüllt:* Versionsverwaltungssysteme werden nicht unterstützt. |

## 4.2 Technische Kriterien

Wie im Kapitel 3.2.2 zum Vorgehen für technische Kriterien beschrieben, werde ich die Kriterien anhand zweier Verfahren erstellen. Zum einen fasst ALARCÓN (vgl. [Ala06]) viele gängige Kriterien aus der Literatur zusammen, aus dem ich solche entnehmen werde, die für Softwaresysteme relevant sind. Zum anderen werde ich Kriterien mittels explorativer Anwendung des bereits genutzten Softwarewerkzeugs aus dem Fallbeispiel, anhand von technischen Eigenschaften gewinnen. Dieses Werkzeug wird im Zuge der Bewertung im Kapitel 5.1 genauer vorgestellt werden. Aufgrund häufiger Überschneidungen der aufgestellten Kriterien mit denen aus der Literatur, werden beide Quellen in einer Kriterienliste zusammengefasst werden.

| **Kriterium** | Plattformunabhängigkeit |
|---|---|
| **Beschreibung** | Das Werkzeug soll auf möglichst vielen Betriebssystemen lauffähig sein. |
| **Motivation** | Höhere Flexibilität. Kosteneinsparung durch die Verwendung von freien Systemen. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Das Werkzeug läuft auf allen relevanten Systemen.<br>• *überwiegend erfüllt:* Das Werkzeug läuft auf den meisten relevanten Systemen.<br>• *teilweise erfüllt:* Das Werkzeug läuft auf wenigen Systemen.<br>• *nicht erfüllt:* Das Werkzeug läuft nur auf einem System. |

| **Kriterium** | Offene und dokumentierte Schnittstellen |
|---|---|
| **Beschreibung** | Das Werkzeug soll die Möglichkeit bieten von außen über standardisierte Schnittstellen auf Datensätze zuzugreifen, sowie das Werkzeug selber zu steuern. |
| **Motivation** | Möchte man beispielsweise automatisiert Statistiken für ein Release generieren, ist eine direkte, umfassende Kommunikation mit dem Werkzeug wünschenswert. |
| **Indikator** | Diese Informationen sind der Dokumentation des Werkzeugs entnehmbar. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug bietet eine allumfassende, standardisierte und dokumentierte Schnittstelle an. Darüber können sämtliche Daten abgefragt und alle Funktionen verwendet werden.<br>• *überwiegend erfüllt:* Das Werkzeug bietet viele Daten und Funktionen über eine Schnittstelle an.<br>• *teilweise erfüllt:* Einige wenige Daten und Funktionen werden nach außen angeboten.<br>• *nicht erfüllt:* Eine Schnittstelle wird vom Werkzeug nicht angeboten. |

| **Kriterium** | Preis/Leistung |
|---|---|
| **Beschreibung** | Das Verhältnis vom Preis zur gegebenen Leistung rechtfertigt den Erwerb. |
| **Motivation** | Software, die ihren Preis für einen bestimmten Einsatzzweck nicht ist, ist unwirtschaftlich. |
| **Indikator** | Die Leistung eines Werkzeugs zu bewerten, ist Teil dieser Arbeit. Ob der Preis die Leistung rechtfertigt liegt außerhalb der Aufgabenstellung und wird hier daher nur grundsätzlich beschrieben, der Vollständigkeit halber aber mit aufgenommen. |
| **Bewertung** | • *voll erfüllt:* Der Preis rechtfertigt die Leistung in vollem Umfang.<br>• *überwiegend erfüllt:* Der Preis rechtfertigt überwiegend die Leistung.<br>• *teilweise erfüllt:* Der Preis rechtfertigt die Leistung nur teilweise.<br>• *nicht erfüllt:* Der Preis rechtfertigt die Leistung in keiner Weise. |

| **Kriterium** | Unterstützung vom Hersteller und anderen Instanzen |
|---|---|
| **Beschreibung** | Der Grad der Unterstützung seitens des Herstellers zu Themen wie beispielsweise Installation, Betrieb oder sonstigen Angelegenheiten. |
| **Motivation** | Fehlendes Know-How kann durch eine Unterstützung ausgeglichen werden. |
| **Indikator** | Der Grad der Unterstützung hängt vom Hersteller und den Vertragsdetails ab. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Der Hersteller leistet für alle Bereiche volle Unterstützung. Zusätzlich wird ein gemeinschafts-orientierter Ansatz in Form eines Forums oder einer Mailingliste zur Verfügung gestellt.<br>• *überwiegend erfüllt:* Der Hersteller leistet für alle Bereiche volle Unterstützung.<br>• *teilweise erfüllt:* Der Hersteller leistet keine Unterstützung, aber es gibt eine aktive Gemeinschaft zu dem Werkzeug.<br>• *nicht erfüllt:* Der Hersteller leistet keine Unterstützung, es gibt auch keine aktive Gemeinschaft zu dem Werkzeug. |

| **Kriterium** | Erweiterbarkeit |
|---|---|
| **Beschreibung** | Das Werkzeug lässt sich auf verschiedenen Ebenen erweitern. Entweder auf einem High-Level Ansatz wie dem Aussehen, eigenen Authentifizierungsmechanismen oder die Anpassung der Vorgehensweisen im Werkzeug auf die eigenen Ansätze, oder auf einem Low-Level Ansatz, wie das Modifizieren des Quelltextes des Werkzeugs. |
| **Motivation** | Als Unternehmen möchte man eventuell das Aussehen des Werkzeugs an das eigene Unternehmensdesign anpassen, bereits bestehende Authentifizierungsmechanismen nutzen und noch viel wichtiger, die eigenen Vorgehensweisen verwenden. Low-Level Erweiterungen können dann nötig sein, wenn fehlende Funktionen nachgerüstet werden müssen, die auch über einen High-Level Ansatz nicht nachrüstbar wären. |
| **Indikator** | Für den High-Level Ansatz sind die Informationen der Dokumentation des Werkzeugs zu entnehmen, für den Low-Level Ansatz entscheidet die Lizenz, bzw. die Vertragsbedingungen darüber. Ist der Quelltext verfügbar, so kann die verwendete Sprache ein weiteres Kriterium darstellen. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Im High-Level Ansatz lassen sich sehr viele Komponenten erweitern, zusätzlich kann der Quelltext frei modifiziert werden.<br>• *überwiegend erfüllt:* Im High-Level Ansatz lassen sich sehr viele Komponenten erweitern, der Quelltext kann aber nicht modifiziert werden.<br>• *teilweise erfüllt:* Im High-Level Ansatz lassen sich nur wenige Komponenten erweitern, der Quelltext ist aber frei modifizierbar.<br>• *nicht erfüllt:* Im High-Level Ansatz lassen sich nur unzureichend viele oder gar keine Komponenten erweitern und der Quelltext kann nicht modifiziert werden. |

| **Kriterium** | Druckmöglichkeit |
|---|---|
| **Beschreibung** | Aus dem Werkzeug heraus sollen nach Möglichkeit sämtliche Objekte in einer ausreichend guten Form druckbar sein. |
| **Motivation** | Manchmal empfiehlt es sich Gegenstände unabhängig von Computern zu diskutieren. Ein Ausdruck des Gegenstands kann als Unterstützung aber dennoch hilfreich sein. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. Zusätzlich kann an einigen Stellen das Drucken getestet werden. |
| **Bewertung** | • *voll erfüllt:* Es lässt sich alles drucken, sowohl in einer Bildschirm-, als auch in einer speziellen Druckansicht.<br>• *überwiegend erfüllt:* Entweder wird das Drucken über die Bildschirm-, oder über die Druckansicht ermöglicht.<br>• *teilweise erfüllt:* Es lassen sich lediglich einige Aspekte des Werkzeugs ausdrucken.<br>• *nicht erfüllt:* Das Drucken wird nicht unterstützt. |

| **Kriterium** | Hilfesystem |
|---|---|
| **Beschreibung** | Vom Werkzeug wird ein direktes Hilfesystem inklusive einer Dokumentation zur Verfügung gestellt. |
| **Motivation** | Sollten akute Unklarheiten bestehen, kann durch ein Hilfesystem schnell Abhilfe geschaffen werden. |
| **Indikator** | Stichprobenartig überprüfen, ob ein Hilfesystem existiert. |
| **Bewertung** | • *voll erfüllt:* Es wird ein komplettes Hilfesystem zur Verfügung gestellt.<br>• *überwiegend erfüllt:* Es wird ein Hilfesystem mit den wichtigsten Themen zur Verfügung gestellt.<br>• *teilweise erfüllt:* Das Hilfesystem besteht aus einer Kurzreferenz der häufigsten Probleme.<br>• *nicht erfüllt:* Es gibt kein Hilfesystem. |

| **Kriterium** | Dokumentation |
|---|---|
| **Beschreibung** | Der Grad der Dokumentation zu einem Werkzeug |
| **Motivation** | Für das Verstehen und Anwenden von einem Werkzeug oder Software im Allgemeinen, ist eine Dokumentation in der Regel unerlässlich. |
| **Indikator** | Überprüfen, welche Dokumentationen der Hersteller bereitstellt. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Es existiert eine komplette Dokumentation mit einer Einführung, Installations-/Upgrade- und Einrichtungsanweisungen, sowie Hinweisen zur effizienten Benutzung. Außerdem werden alle Features erklärt.<br>• *überwiegend erfüllt:* Es existiert eine fast vollständige Dokumentation.<br>• *teilweise erfüllt:* Die bereitgestellte Dokumentation ist lückenhaft.<br>• *nicht erfüllt:* Eine Dokumentation wird nicht angeboten. |

| **Kriterium** | Verschlüsselung |
|---|---|
| **Beschreibung** | Der Zugriff auf das Werkzeug kann unter Einhaltung kryptographischer Sicherheitsstandards erfolgen. |
| **Motivation** | Wenn ein Werkzeug die Möglichkeit bietet aus unsicheren Netzen auf dieses zuzugreifen, so sollte dieser Zugriff aufgrund von Sicherheitsaspekten abgesichert werden. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. Zusätzlich kann bei browserbasierten Werkzeugen die Unterstützung für HTTPS überprüft werden. |
| **Bewertung** | • *voll erfüllt:* Es wird ein standardisiertes Verschlüsselungsverfahren für die Kommunikation und damit dem Zugriff angeboten.<br>• *überwiegend erfüllt: entfällt*<br>• *teilweise erfüllt:* Ein gesicherter Zugriff ist nicht möglich, kann aber durch das Absichern niederer Protokolle gewährleistet werden.<br>• *nicht erfüllt:* Ein gesicherter Zugriff ist nicht möglich. |

| **Kriterium** | Backupfähigkeit |
|---|---|
| **Beschreibung** | Die Daten des Werkzeugs können gesichert werden. |
| **Motivation** | Im Falle eines Hardwareausfalls bleiben die Daten erhalten. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug kann vollständig gesichert werden.<br>• *überwiegend erfüllt:* Die meisten Daten können gesichert werden.<br>• *teilweise erfüllt:* Nur einige wenige Daten können gesichert werden.<br>• *nicht erfüllt:* Ein Backup ist nicht möglich. |

| **Kriterium** | Historie |
|---|---|
| **Beschreibung** | Das Werkzeug bietet für möglichen Aktionen seiner Komponenten ein Historie. |
| **Motivation** | Durch eine Historie kann eine bessere Nachvollziehbarkeit von Abläufen im Werkzeug gewährleistet werden. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. |
| **Bewertung** | • *voll erfüllt:* Eine Historie wird für sämtliche Aktivitäten des Werkzeugs angelegt.<br>• *überwiegend erfüllt:* Die meisten Aktivitäten erzeugen eine Historie.<br>• *teilweise erfüllt:* Nur wenige ausgewählte Aktivitäten erzeugen eine Historie.<br>• *nicht erfüllt:* Eine Historie wird nicht angeboten. |

| **Kriterium** | Mehrbenutzerbetrieb |
|---|---|
| **Beschreibung** | Das Werkzeug unterstützt die Nutzung von mehreren Personen gleichzeitig. |
| **Motivation** | In der agilen Softwareentwicklung wird in einem Team aus mehreren Personen gearbeitet. Jedes Mitglied soll die Möglichkeit haben mit dem Werkzeug zu arbeiten. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. |
| **Bewertung** | • *voll erfüllt:* Beliebig viele Personen können mit dem Werkzeug sowohl technisch, als auch inhaltlich arbeiten. Dazu hat jeder Benutzer eine eigene Kennung.<br>• *überwiegend erfüllt:* Das Werkzeug kann zwar von mehreren Personen gleichzeitig bedient werden, doch fehlt eine semantische Unterscheidung der Nutzer - Es gibt nur einen Benutzer.<br>• *teilweise erfüllt: entfällt*<br>• *nicht erfüllt:* Das Werkzeug ist nur für eine Person ausgelegt. |

| **Kriterium** | Kontrollierbarkeit des Zugangs |
|---|---|
| **Beschreibung** | Der Zugang zum Werkzeug kann durch eine Zugangskontrolle abgesichert werden. |
| **Motivation** | Nur autorisierte Personen dürfen auf das Werkzeug zugreifen. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Jeder Benutzer hat seinen eigenen Account mit seinen eigenen Zugangsdaten.<br>• *überwiegend erfüllt:* Es wird ein globales Passwort für den Zugang gesetzt.<br>• *teilweise erfüllt: entfällt*<br>• *nicht erfüllt:* Das Werkzeug bietet keine Möglichkeit einer Zugangsbeschränkung. |

| **Kriterium** | Sprachunterstützung |
|---|---|
| **Beschreibung** | Verschiedene Sprachen in der das Werkzeug bedient werden kann. |
| **Motivation** | Das Werkzeug sollte nach Möglichkeit die eigene Sprache, die Sprache des Kunden, sowie die englische Sprache unterstützen. Optimal ist es, wenn das Werkzeug ein multilinguales Interface anbietet. |
| **Indikator** | Diese Information ist der Dokumentation des Werkzeugs entnehmbar. |
| **Bewertung** | • *voll erfüllt:* Das Werkzeug ist multilingual.<br>• *überwiegend erfüllt:* Die eigene Sprache, die Sprache des Kunden, sowie die englische Sprache werden unterstützt.<br>• *teilweise erfüllt:* Es wird lediglich die englische Sprache unterstützt.<br>• *nicht erfüllt:* Das Werkzeug benutzt eine unbekannte Sprache. |

| **Kriterium** | Aktivität des Projekts |
|---|---|
| **Beschreibung** | Beschreibt die derzeitige Aktivität des Projekts, in dem das Werkzeug entwickelt und gewartet wird. |

| | |
|---|---|
| **Motivation** | Die Aktivität sollte hoch sein, um mit Neuerungen und Fehlerbereinigungen versorgt werden zu können. |
| **Indikator** | Für diese Feststellung ist am einfachsten die bisherige Versionshistorie des Werkzeugs zu betrachten. An dieser kann man die Frequenz für Updates erkennen, bzw. wie alt die neuste Version ist. |
| **Bewertung** | • *voll erfüllt:* Hohe Frequenz von Updates und Fehlerbereinigungen, sowie eine aktive Entwicklergruppe.<br>• *überwiegend erfüllt:* Updates und Fehlerbereinigungen werden unregelmäßig veröffentlicht.<br>• *teilweise erfüllt:* Fehlerbereinigungen werden zur Verfügung gestellt.<br>• *nicht erfüllt:* Aktivität ist derzeit nicht erkennbar. |

| **Kriterium** | Geschwindigkeit |
|---|---|
| **Beschreibung** | Das Werkzeug soll schnell arbeiten, Wartezeiten sollen vermieden werden. |
| **Motivation** | Wartezeiten unterbrechen den Arbeitsfluss und beeinträchtigen die Konzentration. |
| **Indikator** | Anhand einiger typischer Operationen für das jeweilige Werkzeug soll die Geschwindigkeit gemessen werden. Laut ALARCÓN (vgl. [Ala06]) bedeutet ein rascher Bildaufbau eine Wartezeit von unter 7 Sekunden. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Keine gefühlten Wartezeiten.<br>• *überwiegend erfüllt:* Es treten nur selten längere Wartezeiten auf. Die gefühlte Geschwindigkeit ist insgesamt noch annehmbar.<br>• *teilweise erfüllt:* Es treten ab und zu längere Wartezeiten auf.<br>• *nicht erfüllt:* Generelles Auftreten von Wartezeiten, die mehr als 7 Sekunden betragen. |

## 4.3 Usability-Kriterien

Die Bewertung der Usability erfolgt, wie in Kapitel 3.2.3 zum Vorgehen dargelegt, anhand der Heuristiken von NIELSEN (vgl. [Nie93]) erfolgen. Die Benennung erfolgt im Original in englischer Sprache. Sie lauten im Detail:

| **Kriterium** | Visibility of system status |
|---|---|
| **Beschreibung** | Der Benutzer soll stets über den aktuellen Zustand des Werkzeugs informiert werden. Um dies zu erreichen werden verschiedene Methoden angewandt. Um Aktionen zu verdeutlichen, kann sich beispielsweise der Cursor bei Links in eine Hand ändern. Um Dinge zu verdeutlichen können diese hervorgehoben werden. Informationen können in einer Statusleiste angezeigt angezeigt werden. Zu diesem Kriterium gehören auch Antwortzeiten. Zeiten bis 100 ms entsprechen der gefühlten Echtzeit. |
| **Motivation** | Maximal eine Sekunde kann ein Benutzer warten, bevor dessen Denkprozess unterbrochen wird. Ab zehn Sekunden verliert der Benutzer die Aufmerksamkeit. Abhilfe schaffen Fortschrittsanzeigen. Der Bereich von einer bis zehn Sekunden lässt sich mit dem typischen Sanduhr-Mauszeiger überbrücken. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| | |
|---|---|
| **Kriterium** | Match between system and the real world |
| **Beschreibung** | Für die Interaktion soll die natürliche Sprache verwendet und technische Fachwörter vermieden werden. Dazu zählt natürlich nicht die Verwendung domänespezifischer Fachwörter innerhalb der Anwendung. Wenn Dinge vom Benutzer benannt werden dürfen, soll es keine künstliche Beschränkung für die Länge der Benennung geben. Für die Aufrecherhaltung des Konzepts der echten Welt innerhalb des Systems können Metaphern verwendet werden. Diese können den Benutzer aber auch verwirren. |
| **Motivation** | Wenn man sich mit dem Benutzer nicht auf einer sprachlichen Ebene befindet, kann die Kommunikation scheitern. Ebenso hat der Benutzer auch nicht immer ein Verständnis dafür, dass Eingaben in irgendeiner Form beschränkt sind. Die Verwendung von Metaphern kann das Verständnis des Werkzeugs erleichtern. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | User control and freedom |
|---|---|
| **Beschreibung** | Das Werkzeug soll dem Benutzer keine Entscheidungen aufzwingen. Jeder Dialog, oder auch eine lange Operation, muss eine Möglichkeit zum Abbrechen bieten. Zusätzlich bedarf es einer Funktion Aktionen Rückgängig zu machen. |
| **Motivation** | Zwingt ein System einen Benutzer zu einer Operation verliert er das Vertrauen. Er sollte in der Lage sein ein Interface zu erkunden ohne Befürchten zu müssen etwas nachhaltig verändert zu haben. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Consistency and standards |
|---|---|
| **Beschreibung** | Der Benutzer soll niemals überrascht werden. Ähnliche Dinge sollen sich immer gleich verhalten und auch gleich aussehen. Wenn sich Dinge unterscheiden, sollen diese auch unterschiedlich aussehen und ein anderes Verhalten zeigen. Die verwendete Sprache muss konsistent und einheitlich gehalten werden. Für ein und dieselbe Operation darf es nicht mehrere Bezeichnungen geben. |
| **Motivation** | Nicht konsistente Dinge verwirren den Benutzer und stören ihn im Arbeitsprozess. Wenn der Benutzer zwei unterschiedliche Beschreibungen für eine Operation sieht, wird er auch annehmen, dass es sich im zwei unterschiedliche Operationen handelt. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Error prevention |
|---|---|
| **Beschreibung** | Um Fehlern vorzubeugen sollte man den Benutzer nur das Nötigste selber tippen lassen. Anstelle dessen sollte der Benutzer etwas auswählen. Natürlich ist dies nur bis zu einem gewissem Grad sinnvoll. Wenn eine Aktion zu einem Zeitpunkt oder Zustand nicht erlaubt ist, darf diese dem Benutzer gar nicht erst angezeigt werden, bzw. soll der Button dafür ausgegraut werden. |
| **Motivation** | Der Mensch macht ständig Fehler. Anstatt Fehler zu behandeln ist es am besten Fehler gar nicht erst entstehen zu lassen. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Recognition rather than recall |
|---|---|
| **Beschreibung** | Für das Gehirn ist es einfacher bereits kennengelernte Dinge wiederzuerkennen, als diese selber abzurufen. Das Werkzeug soll daher Menüs o.ä. verwenden, anstatt einer Kommandosprache. Comboboxen[3] sind Textfeldern vorzuziehen. Außerdem sollen generische Operationen angeboten werden und sich auch entsprechend verhalten. Dazu zählen zum Beispiel "Öffnen", "Speichern', "Drucken" sowie "Kopieren" und "Einfügen". Ein weiteres Merkmal ist die Anzeige von allen Informationen, die in einer bestimmten Situation benötigt werden. Der Benutzer soll sich diese nicht merken müssen. |
| **Motivation** | Dieses Kriterium minimiert die geforderte Leistung des Gehirns. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Flexibility and efficiency of use |
|---|---|
| **Beschreibung** | Das Werkzeug soll Abkürzungen für besonders häufig auszuführende Operationen anbieten. Dazu zählen Tastenkombinationen, personalisiertes Aussehen, Lesezeichen und eine Historie der ausgeführten Aktionen. |
| **Motivation** | Erfahrenen Benutzern wird die Möglichkeit des schnelleren Bedienens und damit des effizienteren Arbeitens gegeben. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben.. |

[3] Ein GUI-Widget, welches eine Liste von Optionen anzeigt, aus denen der Benutzer wählen kann.

| | |
|---|---|
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Aesthetic and minimalist design |
|---|---|
| **Beschreibung** | Das Werkzeug sollte hinsichtlich des Designs simpel gestaltet sein. Farben sollten mit Bedacht und anhand von einigen Grundsätzen für die Farbwahl und Schriftartwahl verwendet werden. Wenn Dinge überflüssig sind, sollen sie auch keinen Platz wegnehmen. |
| **Motivation** | Durch dieses Kriterium wird eine bessere Übersichtlichkeit gefördert. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Help users recognize, diagnose, and recover from errors |
|---|---|
| **Beschreibung** | Falls ein Fehler auftritt, soll möglichst viel getan werden, um den Benutzer bei der Behebung zu unterstützen. Dazu gehören präzise Angaben über den Fehler, das Sprechen der Sprache des Benutzers, sowie das Ausblenden technischer Details (es sei denn sie werden explizit gefordert) und konstruktiv Hinweise für die Behebung. Dabei soll das System dem Benutzer gegenüber stets freundlich sein. |

| | |
|---|---|
| **Motivation** | Eine Fehlermeldung mit technischen Details hilft dem Benutzer nicht. Zudem könnte es die Gefühle des Benutzers negativ beeinflussen, wenn keinerlei Hinweise für die Fehlerbehebung mit angegeben werden. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

| **Kriterium** | Help and documentation |
|---|---|
| **Beschreibung** | Normalerweise werden Handbücher von Benutzern nur dann gefordert und gelesen, wenn sie an irgendeinem Punkt nicht weiterkommen. Daher sollte ein gutes Handbuch durchsuchbar sein, Kontext-sensitiv arbeiten und konkrete und kurze Hinweise für eine Lösung geben. |
| **Motivation** | Das Handbuch kann die erste Anlaufstelle sein, wenn der Benutzer nicht weiterkommt. Dieses muss entsprechend der Problematik auf den Benutzer eingehen können, sonst hat es für ihn keinen Wert. |
| **Indikator** | Wie im Vorgehen in Kapitel 3.2.3 beschrieben. |
| **Bewertung** | • *voll erfüllt:* Keine Beanstandung gefunden.<br>• *überwiegend erfüllt:* Eine Beanstandung gefunden.<br>• *teilweise erfüllt:* Zwei Beanstandungen gefunden.<br>• *nicht erfüllt:* Drei oder mehr Beanstandungen gefunden. |

# 5 Bewertung

Nachdem der Kriterienkatalog erstellt wurde, soll er in diesem Kapitel exemplarisch auf zwei Werkzeuge angewendet werden. Dazu werden die beiden Kandidaten zunächst vorgestellt und hinsichtlich ihrer Zielsetzung charakterisiert. Anschließend findet die Bewertung statt.

## 5.1 Kandidaten vorstellen

Der erste Kandidat ist *XPlanner*[1]. *XPlanner* ist ein freies, browserbasiertes Werkzeug zum Planen und Verfolgen der agilen Softwareentwicklung, insbesondere von "extreme Programming" (XP). Für die Bewertung liegt es zum Zeitpunkt der Arbeit in der aktuellsten Version *0.7b7* vor. Es kann kostenlos von der Projektseite bezogen werden. Das Konzept der Anwendung beruht auf Storycards (Abbildungen 5.1 und 5.2), Tasks und Iterationen. Darüber hinaus werden unterschiedliche Metriken anhand der gegebenen Daten generiert und dem Benutzer in Form von Statistiken und Grafiken (Abbildungen 5.3 und 5.4) zur Verfügung gestellt. Die aufgeführten Grafiken sollen einen Eindruck von *XPlanner* vermitteln.


Abbildung 5.1: Story in XPlanner

[1]Projektseite: http://www.xplanner.org

**Stunden: Schätzung 143,3, Verbrauchte 134,0, Verbleibend 9,3**

Save order

| Actions | ID | Order | User Story | ! | Cust. | Progress | Rem. | Cur. Est. | Orig. Est. | Tasks | Tracker | Disp. | Status |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 31077 | 1 | WZ Vertrags-Editor: Generische Steuerung für Bedingungen/PV-Ergänzungen | 1 | KRE | 26,8 | | 26,8 | 18,0 | 8 | GAB | Planned | Verified |
| | 55760 | 2 | Performance-Aktivitäten / Analyse PUFFER | 4 | | 0,0 | 5,5 | 5,5 | 6,5 | 1 | | Planned | Planned |
| | 23827 | 3 | Einbau der Multi-Document-Verwaltung | 2 | | 25,1 | | 25,1 | 17,5 | 9 | | Planned | Verified |
| | 34209 | 4 | Einbau der Multi-Document-Verwaltung II | 2 | | 7,6 | | 7,6 | 7,2 | 6 | | Planned | Verified |
| | 54982 | 5 | Umgestaltung der Länderliste | 4 | KRE | 18,2 | | 18,2 | 9,0 | 2 | SEL | Planned | Verified |
| | 40965 | 6 | VordeklarationsWerkzeug: Sequenznummer nicht als Index verwenden | 4 | | 5,3 | | 5,3 | 2,5 | 4 | | Planned | Verified |
| | 55461 | 7 | Performance-Aktivitäten / Programmierung PUFFER | 4 | | 0,0 | | 0,0 | 7,0 | 1 | | Planned | Planned |
| | 61671 | 8 | [Performance] Session-Cache von Hibernate | 3 | SEL | 3,4 | | 3,4 | 4,0 | 1 | GAB | Planned | Verified |
| | 61668 | 9 | [Performance] Optimierung der Anfragen an die DB | 2 | | 1,0 | | 1,0 | 4,0 | 1 | GAB | Planned | Verified |
| | 44280 | 10 | B: VD Warten bei Sortierung Waren und Leistungen | 4 | SEL | 2,9 | | 2,9 | 3,0 | 1 | | Planned | Verified |
| | 55761 | 11 | Bugfixing | 4 | | 8,2 | 3,8 | 12,0 | 12,0 | 3 | | Planned | Planned |
| | 61674 | 12 | Historie der Entgeltsätze bei APG-light | 2 | KRE | 8,6 | | 8,6 | 4,0 | 1 | GAB | Planned | Verified |
| | 61669 | 13 | Beim Zuordnen einer VD auch Waren und Leistungen kopieren | 2 | KRE | 1,2 | | 1,2 | 3,0 | 1 | | Planned | Verified |
| | 57718 | 14 | P: B: ReferenceDataService liefert leere Listen | 2 | | 2,8 | 0,0 | 2,8 | 2,8 | 1 | | Planned | Planned |
| | 54150 | 15 | Converter für ReferenceData schreiben | 1 | | 2,8 | | 2,8 | 3,5 | 3 | | Planned | Verified |
| | 60769 | 16 | ReferenceData an EAP Vorlage anpassen | 4 | SRE | 4,4 | | 4,4 | 4,5 | 2 | | Planned | Verified |
| | 57440 | 17 | EAP-AAA-Template | 4 | | 2,7 | | 2,7 | 0,0 | 1 | | Added | Verified |
| | 61982 | 18 | [Perfomance] Optimierung: Fetching für anhängige Listen konfigurieren | 1 | | 13,1 | | 13,1 | 0,0 | 1 | | Added | Verified |

Save order

Bearbeiten | Neue User Story | Beenden | Import | Export | Stories | Alle Tasks | Metriken | Statistiken | Genauigkeit | Historie | Drucken

Abbildung 5.2: Übersicht aller Stories einer Iteration in XPlanner

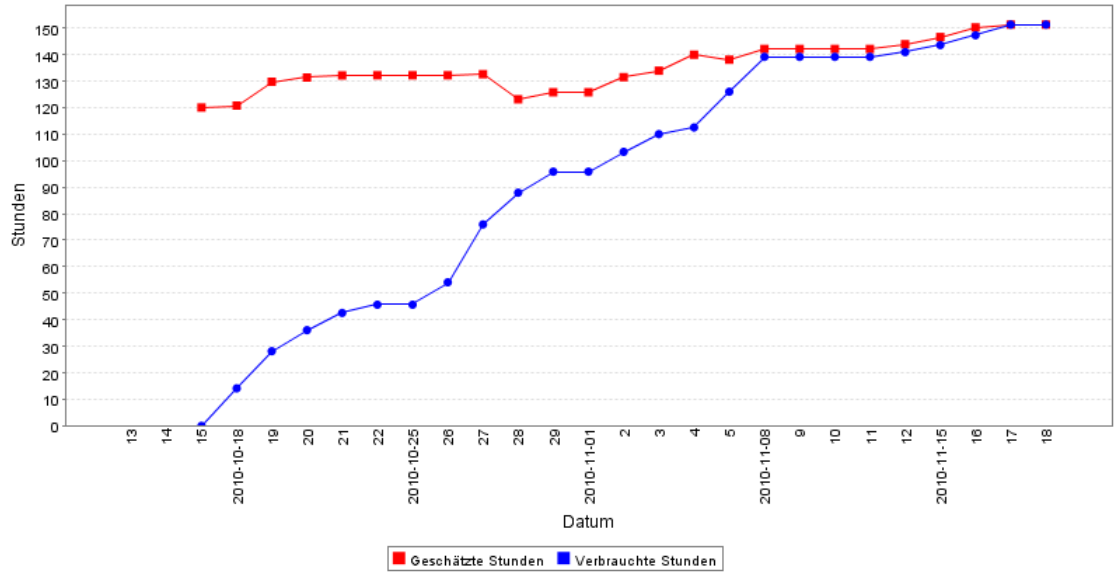


Abbildung 5.3: Statistik über verbrauchte Stunden in XPlanner


Abbildung 5.4: Statistik über die Anzahl von bestimmten Typen von Stories in XPlanner

*XPlanner* ist das Werkzeug, welches derzeit als unterstützendes Werkzeug im Fallbeispiel verwendet wird.

Der zweite Kanditat ist *Mingle*.[2] *Mingle* ist ein kommerzielles, ebenfalls browserbasiertes Werkzeug zur Projektverwaltung agiler Softwareentwicklung, welches sich das Ziel gesetzt hat, sich auf bestehende Vorgehensweisen anpassen zu können. Allerdings ist *Mingle* aufgrund des kommerziellen Hintergrunds nicht kostenlos. Zum Testen wird aber vom Hersteller (”ThoughtWorks Studios”) eine kostenlose, dafür zeitlich beschränkte und auf fünf Benutzer beschränkte Version angeboten. Für die Bewertung liegt es zum Zeitpunkt der Arbeit in der aktuellsten Version *3.2* vor. Anders als bei *XPlanner* werden keine starren Vorgehensweise verwendet. Das Konzept sieht eine hohe Konfigurierbarkeit vor. Alle Entitäten sind zunächst sogenannte ”Cards”. Diese können in Abhängigkeit zueinander gesetzt werden, unterschiedliche Attribute besitzen und so die Form von Iterationen, Stories oder Tasks annehmen. Daher wird *Mingle* für die Bewertung anhand der in Kapitel 4.1 beschriebenen Szenarien entprechend konfiguriert. Um ebenfalls einen Eindruck über *Mingle* zu gewinnen, folgen einige Grafiken, in denen Stories (Abbildungen 5.5 und 5.6) gezeigt werden, sowie Beispiele für Statistiken (Abbildungen 5.7 und

[2]Herstellerseite: http://www.thoughtworks-studios.com/mingle-agile-project-management

5.8). Die generierten Statistiken lassen sich sowohl textlich, als auch bildlich mithilfe einer Markup-Sprache anpassen.

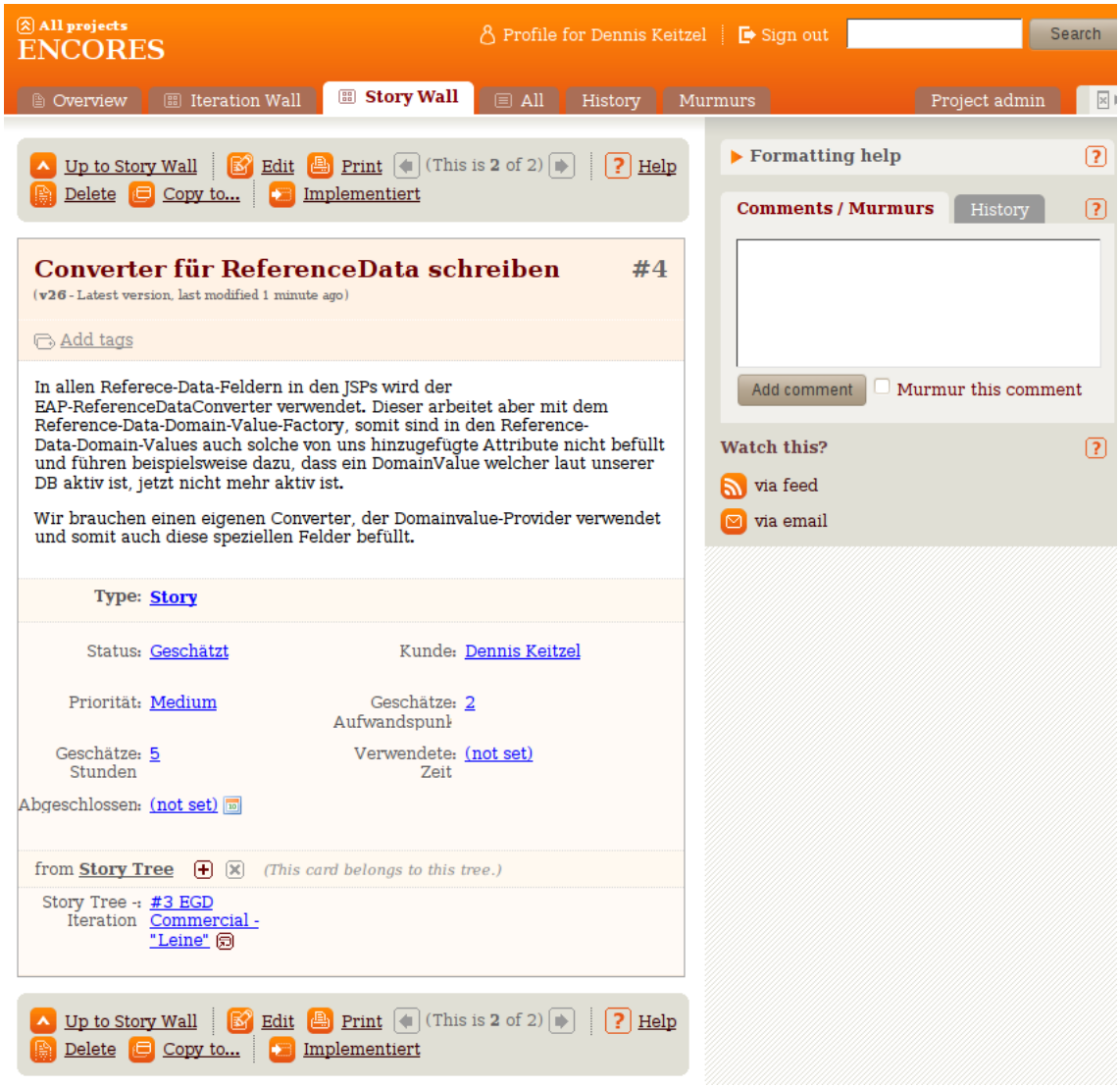


Abbildung 5.5: Story in Mingle

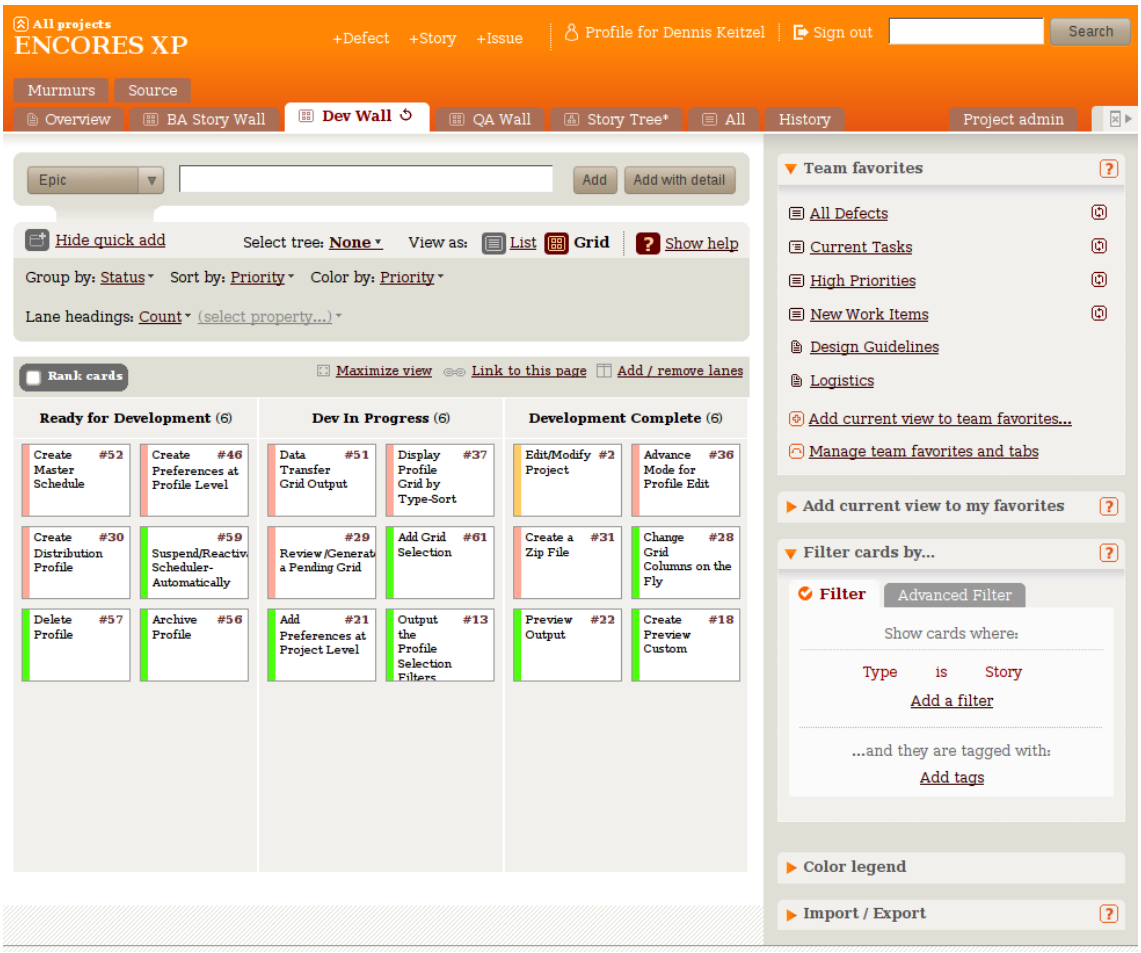


Abbildung 5.6: Übersicht der Stories, die derzeit für den Entwickler relevant sind

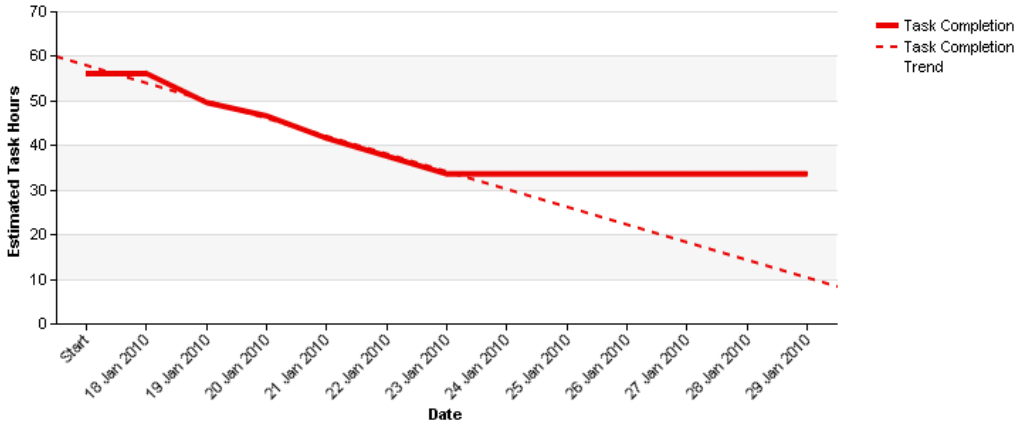


Abbildung 5.7: Burn-Down Statistik in Mingle

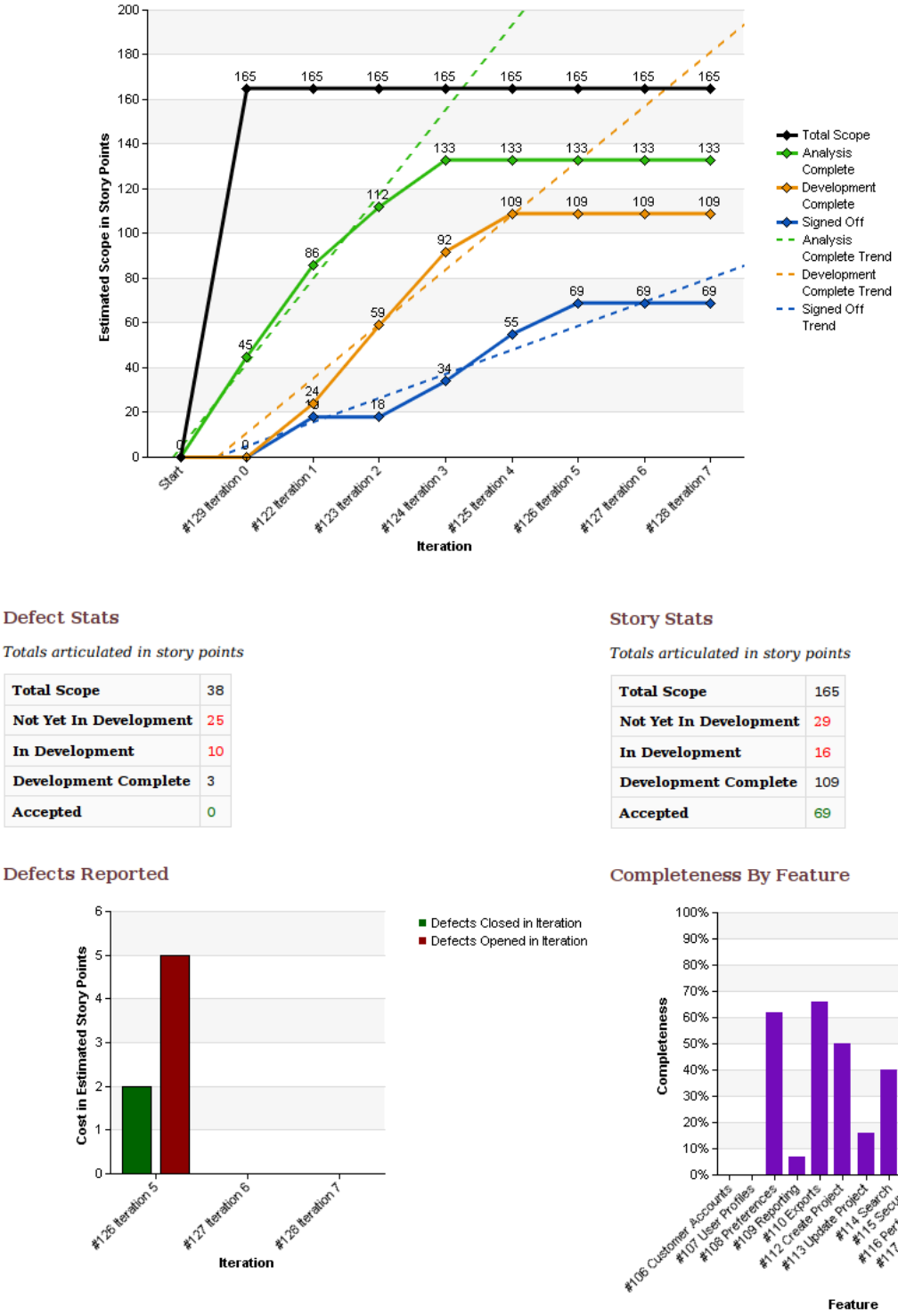


Abbildung 5.8: Beispiele für Statistiken in Mingle

## 5.2 Bewertung durchführen

Die Bewertung erfolgt in einer Tabelle. Es werden jeweils nur die Namen der Kriterien angegeben. Für eine ausführliche Beschreibung und das jeweilige Bewertungsschema

sei auf das Kapitel 4 zum Kriterienkatalog verwiesen. Die erreichte Punktzahl wird für jeden Kandidaten in einer zweiten und dritten Spalte angegeben. Anmerkungen zu den Bewertungen erfolgen über Fußnoten.

| **Fachliche Kriterien** | **XPlanner** | **Mingle** |
|---|---|---|
| Unterstützung des Story-Konzepts | 2 | 3 |
| Schätzen in relativen Aufwandspunkten und absoluten Zeiteinheiten | 1 | 3 |
| Stories & Tasks haben Zustände | 2 | 3 |
| Zuordbarkeit von Personen zu Stories | 2 [3] | 1 [4] |
| Die Möglichkeit Tasks als hierarchische Aufgabenteilung für Stories zu verwenden | 2 | 3 |
| Kommentierbarkeit von Stories | 2 | 1 |
| Priorisierbarkeit von Stories | 1 | 3 |
| Ablageplätze für Stories, unabhängig vom Status | 1 | 1 [5] |
| Definierbakeit von Iterationen | 2 | 1 [6] |
| Zuordbarkeit von Stories zu Iterationen | 3 | 2 |
| Kennzeichenbarkeit von Slackstories | 1 | 3 |
| Definierbarkeit der Reihenfolge von Stories einer Iteration | 1 | 1 |
| Definierbarkeit von Abhängigkeiten der Stories einer Iteration | 0 | 0 |
| Dokumentierbarkeit der verwendeten Zeit | 2 | 1 |
| Visualisierbarkeit des Fortschritts einer Iteration | 2 | 3 |
| Definierbarkeit von speziellen Typen von Stories | 0 | 3 |
| Der Ort einer Story kann beliebig wechseln | 3 | 3 |
| Aufteilbarkeit von Stories | 3 | 1 |

[3] Es lassen sich zwar nicht mehrere Personen einem Attribut zuordnen, doch können mehrere Person Arbeitszeit auf einer Story buchen.

[4] Es lassen sich nicht mehrere Personen einem Attribut zuordnen.

[5] Es lassen sich zwar Orte anlegen, doch lassen sich die dort zugeordneten Stories nicht mehr sinnvoll handhaben, um sie später einer Iteration zuordnen zu können.

[6] Eine besondere semantische Bedeutung haben Iterationen nicht. Insbesondere lassen sie sich nicht starten und stoppen. Es bestünde aber die Möglichkeit dieses Verhalten ansatzweise zu modellieren.

| | | |
|---|---|---|
| Reflektierbarkeit über Aufwandspunkte | 2 | 1 |
| Dokumentierfähigkeit | 1 [7] | 0 |
| Kommunikation mit dem Kunden | 0 | 0 |
| Visualisierbarkeit des gesamten Projektfortschritts | 2 | 3 |
| Bugtracking | 1 | 2 |
| Unterstützung von Tests auf multiplen Ebenen | 0 | 1 |
| Templateunterstützung | 0 | 2 |
| Referenzierbarkeit von Quelltexten | 0 | 1 [8] |
| **Summe:** | 36 | 46 |

Tabelle 5.2: Bewertung der fachlichen Kriterien

| **Technische Kriterien** | **XPlanner** | **Mingle** |
|---|---|---|
| Plattformunabhängigkeit | 3 | 3 |
| Offene und dokumentierte Schnittstellen | 2 | 3 |
| Preis/Leistung[9] | - | - |
| Unterstützung vom Hersteller und anderen Instanzen | 1 | 3 [10] |
| Erweiterbarkeit | 1 | 2 |
| Druckmöglichkeit | 3 | 3 |
| Hilfesystem | 0 | 3 |
| Dokumentation | 2 | 3 |
| Verschlüsselung | 3 | 1 |
| Backupfähigkeit | 3 [11] | 2 [12] |
| Historie | 2 | 3 |
| Mehrbenutzerbetrieb | 3 | 3 |

[7] Es wird ein Wiki angeboten.
[8] Es ist allerdings über Commit-Kommentare möglich, Stories zu referenzieren.
[9] Die Bewertung, ob die Leistung einen Preis rechtfertigt, liegt außerhalb dieser Arbeit.
[10] Durch die kommerzielle Vermarktung bevorteilt.
[11] Durch Sicherung der Datenbank.
[12] Projekte können komplett als Datei exportiert werden.

| | | |
|---|---|---|
| Kontrollierbarkeit des Zugangs | 3 | 3 |
| Sprachunterstützung | 3 | 1 |
| Aktivität des Projekts | 0 | 3 |
| Geschwindigkeit | 2 | 2 |
| **Summe:** | 31 | 38 |

Tabelle 5.4: Bewertung der technischen Kriterien

| **Usability-Kriterien** | **XPlanner** | **Mingle** |
|---|---|---|
| Visibility of system status | 1 [13] | 2 [14] |
| Match between system and the real world | 2 [15] | 3 |
| User control and freedom | 1 [16] | 1 [17] |
| Consistency and standards | 1 [18] | 2 [19] |
| Error prevention | 1 [20] | 3 |
| Recognition rather than recall | 2 [21] | 3 |
| Flexibility and efficiency of use | 0 [22] | 2 [23] |
| Aesthetic and minimalist design | 2 [24] | 2 [25] |

[13] Die Suche benötigt, abhängig vom Suchwort, über zehn Sekunden. Der Fortschritt der Suche wird aber nicht angezeigt. Das Verhalten von klickbaren Icons ist aufgrund fehlender Tooltips oder Beschreibungen oft nicht klar.

[14] Wartezeiten von über einer Sekunde sind die Regel. In vielen Fällen wird dies über eine Art Sanduhr verdeutlicht. Dies ist aber nicht immer der Fall. Das Erstellen von Projekten dauert ungefähr 20 Sekunden, ohne dass der Fortschritt verdeutlicht wird.

[15] Es ist nicht immer erkennbar, was Zahlen in Auflistungen bedeuten.

[16] Es fehlt eine Vorschau für neue Iterationen, Stories und Tasks - Dies ist vor allem für Texte wichtig, die mithilfe einer Mark-up Sprache geschrieben wurden. Aktionen lassen sich nicht rückgängig machen.

[17] Aktionen lassen sich nicht rückgängig machen. Kommentare lassen sich weder bearbeiten noch löschen.

[18] Die Stundenbruchteile von Stundenangaben werden uneinheitlich mit einem Komma oder einem Punkt getrennt. Eine Fehleingabe führt teilweise dazu, dass ein Trennzeichen als Tausendertrennung interpretiert wird, anstatt als Bruchteil. Es ist teilweise nicht klar, was Buttons für Auswirkungen haben. So werden beim Verschieben von Stories die beiden Buttons "Verschieben" und "Weiter" angeboten. Der Unterschied wird nicht klar.

[19] Tooltips für Icons oder Links fehlen teilweise.

[20] Wird für die geschätzte Zeit eines Tasks die Stunde mit einem Komma anstatt einem Punkt getrennt, wird der Wert einfach ignoriert. Der Benutzer wird darauf nicht aufmerksam gemacht. Keine Unterstützung für Default-Werte bei Eingaben und Auswahlmöglichkeiten.

[21] Keine Formatierungshilfe für die Mark-Up-Sprache von Texten, nur eine Dokumentation.

[22] Keine Tastenkombinationen. Kein personalisiertes Aussehen. Keine Lesezeichen oder Abkürzungen (Beobachten von Entitäten).

[23] Keine Tastenkombinationen.

[24] Das Design ist eher funktional und wirkt veraltet.

[25] Das Design wirkt etwas überladen, die einzelen Seiten bieten zu viele Interaktionsmöglichkeiten.

| | | |
|---|---|---|
| Help users recognize, diagnose, and recover from errors | 2 [26] | 2 [27] |
| Help and documentation | 1 [28] | 3 |
| **Summe:** | 13 | 23 |

Tabelle 5.6: Bewertung der Usability

## 5.3 Endergebnis

| **Kategorie** | **XPlanner** | **Mingle** | von maximal |
|---|---|---|---|
| Fachliche Kriterien | 36 | 46 | 78 |
| Technische Kriterien | 31 | 38 | 48 |
| Usability-Kriterien | 13 | 23 | 30 |
| **Summe:** | 80 | 107 | 156 |

Tabelle 5.8: Endergebnis der Bewertung

[26] Fehler bei Eingaben werden teilweise ignoriert und es wird ein Default-Wert gesetzt.
[27] Die Eingabe falscher Datentypen wird zwar bemängelt, doch wird teilweise keine Hilfestellung dazu gegegeben, wie die Eingabe auszusehen hat.
[28] Keine kontextsensitive Hilfe. Dokumentation unvollständig.

# 6 Fazit

Durch das Anwenden der Kriterien des Kriterienkatalogs auf das Fallbeispiel sind bei beiden Kandidaten einige Mängel aufgedeckt worden. Nur wenige dieser Mängel waren wirklich schwerwiegend, doch mussten sie natürlich bei der Bewertung mit berücksichtig werden. Wie aus Tabelle 5.8 der aufsummierten Ergebnisse hervorgeht, hat *Mingle* in jeder Klasse mehr Punkte als *XPlanner* erhalten und liegt dadurch auch insgesamt (*XPlanner*: 80 Punkte; *Mingle*: 107 Punkte) deutlich vorne.

Für eine Interpretation dieses Ergebnisses ist allerdings der Hintergrund der vorliegenden Arbeit zu berücksichtigen. Die Verwendung eines Kriterienkatalogs, der nur von einer Person zusammengestellt wurde, lässt die Frage aufkommen, ob die Auswahl der Kriterien überhaupt vollständig ist. Beim Extrahieren der fachlichen Kriterien können Anforderungen übersehen worden sein. Ebenso kann es technische Kriterien geben, die im Katalog fehlen. Außerdem wurde im Fallbeispiel *XPlanner* bereits im Vorfeld zur Entwicklung eingesetzt. Dadurch stand das Vorgehen der Entwicklung bereits unter dem Einfluss von *XPlanner*. Eventuell würden einige Aspekte der Softwareentwicklung anders verlaufen, wenn ein anderes oder garkein Werkzeug verwendet worden wäre. Ein weiterer wichtiger Punkt ist, dass *Mingle* deutlich mehr Features bietet, als in der Bewertung berücksichtigt wurde. Möglicherweise würden sich bei der Verwendung von *Mingle* auch einige Vorgehensweisen ändern, um den Entwicklungsprozess insgesamt besser zu unterstützen. Auf der anderen Seite wird im Laufe der Zusammenstellung des Kriterienkatalogs überhaupt erst klar, welche Funktionen wirklich benötigt werden. Dadurch wird vermieden sich von Versprechen eines Herstellers blenden zu lassen und es kann sachlicher und objektiver bewerten werden.

Zusätzlich sollte berücksichtigt werden, dass ebenfalls nur eine Person den Kriterienkatalog angewendet und die Bewertung durchgeführt hat. Solche Bewertungen können immer durch Subjektivität beeinflusst werden, auch wenn dies unterbewusst und nicht beabsichtigt stattfindet. Vor allem für die Klasse der Usability-Kriterien ist ein einziger Evaluator nicht ausreichend. Wie im Kapitel 3.2.3 zum Vorgehen bereits ausgeführt, hat NIELSEN (vgl. [Nie93]) über sechs Studien herausgefunden, dass ein einziger Evaluator im Schnitt nur 35% der Usababilityprobleme aufdeckt. Drei bis fünf Personen sollen einen vernünftigen Kompromiss darstellen.

Insgesamt ist eine solche Bewertung immer auch ein Trade-off zwischen dem zur Verfügung stehendem Aufwand und der Qualität der Ergebnisse. Sind ausreichend Mittel in Form von Zeit und Evaluatoren vorhanden, kann eine höhere Qualität der Ergebnisse dadurch erreicht werden, dass mehrere Personen die Kriterien zusammenstellen und auch mehrere Personen die Bewertung unabhängig voneinander durchführen. Die Frage dabei ist, wie viele Mittel für ein ausreichend gutes Ergebnis zur Verfügung gestellt werden müssen.

Die hier durchgeführte Bewertung kann aufgrund gegebener Restriktionen (keine Finanzmittel und damit geringe Anzahl von Evaluatoren, zeitliche Beschränkungen) nur einen Teil eines optimalen Ergebnisses bzw. einer endgültigen Bewertung darstellen. Viel mehr sollte die Herangehensweise an die Fragestellung, wie unterstützende Werkzeuge zur agilen Softwareentwicklung bewertet werden können, entwickelt werden. Genau dieser Aspekt wurde ausführlich in den Kapitel 3 und 4 erarbeitet, indem zunächst geklärt wurde, welche allgemeinen Möglichkeiten einer Bewertung zur Verfügung stehen, und nach Festlegung des Kriterienkatalogs, anhand welcher Überlegungen der Kriterienkatalog zusammengestellt werden kann.

Das Ergebnis dieser Arbeit identifiziert Mingle als tendenziell geeigneteres Werkzeug für die Unterstützung der agilen Softwareentwicklung im vorliegenden Fallbeispiel. Die differenzierte Betrachtung der Ergebnisse macht aber ebenfalls deutlich, dass die Bewertung eines Werkzeugs in verschiedenster Weise erweiterbar ist. Beispielsweise könnte die Zusammenstellung der Kriterien, sowie die anschließende Bewertung, erneut mit mehreren Personen durchgeführt werden. Sollte dies gemeinsam mit den Mitarbeitern aus dem Fallbeispiel geschehen, kann auch über die Anwendung von Gewichtungen nachgedacht werden. Einige Kriterien sind für das konkrete Fallbeispiel sicher höher zu gewichten. Darüber hinaus wird es Kriterien geben, die so wichtig sind, dass sie essentiell für die Auswahl eines Werkzeugs sind. Beispielsweise bietet *Mingle* nicht die Möglichkeit einer ausführlichen Zeiterfassung für Stories und Tasks, geschweige denn für das Programmieren im Paar. Diese Funktion wird im Fallbeispiel derzeit bei *XPlanner* ausführlich genutzt, um über eigene Schätzungen reflektieren zu können. Neben dem Aspekt der Zeiterfassung dürfen auch die Kosten des Werkzeugs nicht außer acht gelassen werden. Während *XPlanner* freie Software ist und kostenlos bezogen werden kann, muss *Mingle* entweder gekauft oder gemietet werden. Nach Anfrage beim Hersteller (”ThoughtWorks Studios”) belaufen sich die Kosten im Falle des Mietens auf 566 $ pro Benutzer pro Jahr und im Falle des Erwerbs auf 995 $ pro Benutzer plus 199 $ pro Benutzer pro Jahr für Unterstützung und neue Versionen (im Falle des Mietens mit enthalten). Diese Kosten müssen gerechtfertigt sein und finanziert werden können.

Nachdem der Kandidat *Mingle* als besser geeignet bewertet wurde, wäre der nächste Schritt ein testweiser Einsatz. Für eine realistische Benutzung im Kontext des Fallbeispiels muss *Mingle* entsprechend der Vorgehensweisen angepasst werden. Zusätzlich müssen einige Testdaten in Form von Iterationen, Stories, Tasks und Benutzern aus der bisherigen *XPlanner*-Produktivumgebung exportiert und in *Mingle* importiert werden. Dies kann zunächst manuell erfolgen. Wird in Betracht gezogen, *XPlanner* später wirklich durch *Mingle* zu ersetzen, so ist aufgrund laufender Projekte und einem großen Bestand an Datensätzen eine automatisierte Portierung der Daten unumgänglich.

Die Arbeit hat gezeigt, dass die Bewertung eines Werkzeugs zur Unterstützung agiler Softwareprozesse vermeintlich leicht umzusetzen ist, im Detail aber doch Aspekte, wie die Zusammensetzung des Kriterienkatalogs oder die Art der Anwendung der Kriterien,

ausschlaggebend sind für das Ergebnis der Bewertung. Mit der verwendeten Methode sind aber durchaus Ergebnisse erreichbar, die eine Entscheidung für oder gegen ein solches Werkzeug fundiert unterstützen können.

# 7 Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen, als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internetquellen - benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit der Einstellung der Bachelor-Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden.

Hamburg, den 20. Dezember 2010

DENNIS KEITZEL

# Literaturverzeichnis


[Ala06] ALARCÓN, Marcela Ximena V.: *Evaluation multimedialer Lernprogramme nach neuropsychologischen und konstruktivistischen Anforderungen des Lernens*, Institut für Sprache und Kommunikation der Technischen Universität Berlin, Diss., 2006

[BA04] BECK, Kent ; ANDRES, Cynthia: *Extreme Programming Explained : Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004

[Bal09] BALZERT, Helmut: *Lehrbuch der Software-Technik: Basiskonzepte und Requirements Engineering*. 3. Heidelberg : Spektrum, 2009

[Bau97] BAUMGARTNER, Peter: Didaktische Anforderungen an (multimediale) Lernsoftware. In: *Information und Lernen mit Multimedia* 2. Auflage (1997), S. 241–252

[FMP08] FRANKE, Stefan ; MÜLLER, Christoph ; PRZYBYLSKI, Diana: *Analyse und Kritik von Anforderungsspezifikationen*, Institut für Softwaretechnologie, Universität Stuttgart, Fachstudie, 2008

[HF01] HIGHSMITH, Jim ; FOWLER, Martin: The Agile Manifesto. In: *Software Development Magazine* 9 (2001), Nr. 8, S. 29–30

[Hol03] HOLZINGER, Andreas: *Beurteilungskriterien für Lernsoftware*. Universität Graz. `http://user.meduni-graz.at/andreas.holzinger/holzinger%20de/papers%20de/Beurteilung_Lernsoftware.pdf`. Version: 2003. – URL geprüft am 18.12.2010

[Hol05] HOLZINGER, Andreas: Usability engineering methods for software developers. In: *Commun. ACM* 48 (2005), January, S. 71–74

[Mei95] MEIER, Anne: *Wer braucht Kriterienkataloge?* 1. Bildung und Wissen, 1995. – S. 150-191

[Nie93] NIELSEN, Jakob: *Usability Engineering*. San Francisco, California : Morgan Kaufmann Publishers, 1993

[OMRK92] OPPERMANN, Reinhard ; MURCHNER, Bernd ; REITERER, Harald ; KOCH, Manfred: *Software-ergonomische Evaluation - Der Leitfaden EVADIS II*. Walter de Gruyter, 1992

# Abbildungsverzeichnis