Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Quantitativer und qualitativer Vergleich von Anforderungen bei agilen und konventionellen Softwareprojekten

Studienarbeit

im Studiengang Mathematik mit Studienrichtung Informatik

von

Melanie Hennemann

Prüfer: Prof. Dr. Kurt Schneider
Betreuer: M. Sc. Eric Knauss

Hannover, 15. Juli 2008

Danksagung

Hiermit möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Besonderer Dank gilt M. Sc. Eric Knauss für die hervorragende Betreuung meiner Studienarbeit.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Jede Softwareentwicklung basiert auf den Anforderungen an das System. Die Anforderungen spiegeln die Wünsche des Kunden wieder. Um seine Erwartungen halten zu können, muss die Erfüllung der Anforderungen bei der Entwicklung an oberster Stelle stehen. Jede Methode zur Softwareentwicklung hat eine eigene Art, mit Anforderungen umzugehen.

Methoden zur Softwareentwicklung werden im Wesentlichen in zwei Arten unterteilt: konventionelle und agile Methoden (z.B. eXtreme Programming). Die von Boehm aufgestellte Agilitätsskala vergleicht Softwareentwicklung auf Basis des Planungshorizonts und zeigt, dass agile Projekte nicht so detailliert vorausplanen wie konventionelle (vgl. [Boeh02]). Agile Methoden erfordern also einen anderen Umgang mit Anforderungen als konventionelle Methoden.

Ein wesentlicher Unterschied zum Umgang mit Anforderungen bei konventionellen und agilen Projekten zeigt sich darin, dass bei konventionellen Projekten die Anforderungen ausführlich und verbindlich dokumentiert werden, während bei agilen Methoden wie z.B. eXtreme Programming die Anforderungen nur stichwortartig festgehalten, aber bei Bedarf vom On-Site-Customer erläutert werden. So können beim eXtreme Programming (XP) Anforderungen schnell und unkompliziert geändert werden.

Um sich nun im Vorfeld für die richtige Softwareentwicklungs-Methode entscheiden zu können, ist es doch interessant zu sehen, welche Effekte der unterschiedliche Umgang mit Anforderungen mit sich bringt und wie man sich die Vorteile beider Methoden zunutze machen kann, um die Produktentwicklung zu optimieren.

1.2 Ziele dieser Arbeit

So kann vermutet werden, dass durch die bei XP möglichen nachträglichen Änderungen von Anforderungen und den direkten Kontakt mit dem Kunden vor Ort mehr über nicht-funktionale als über funktionale Anforderungen und mehr über Details gesprochen wird. Dem Kunden wird bei XP schon früh brauchbare Software geliefert. Das macht es ihm leichter, genaue Vorstellungen von seinem gewünschten Endprodukt zu entwickeln und dank der guten Kommunikationsmöglichkeiten und den einkalkulierten Änderungen diese den Entwicklern auch zu vermitteln. Diese Vorstellungen des Kunden werden vermutlich mit dem Fortschritt des Projekts immer präziser werden und gezielt die Eigenschaften der Software beschreiben. Außerdem ist es sehr wahrscheinlich, dass die Möglichkeit, Anforderungen nachträglich zu ändern bei der Mehrzahl der Anforderungen tatsächlich in Anspruch genommen wird. Diese naheliegenden Annahmen gilt es in dieser Arbeit zu überprüfen.

1.3 Aufgabenstellung

Diese Arbeit untersucht den Effekt der unterschiedlichen Herangehensweise an Softwareentwicklung anhand von Messungen. Zum einen soll die Herangehensweise mit agilen Vorgehensmodellen und zum anderen die mit konventionellen Vorgehensmodellen analysiert werden. Anschließend wird man beide Herangehensweisen vergleichen können.

Die Messung der agilen Softwareentwicklung soll nach der Goal-Question-Metric (GQM) Methode innerhalb der Laborübung „Agile Softwareentwicklung“ durchgeführt werden. Bei der Messung der konventionellen Softwareentwicklung stehen bis zu 40 teilweise schon ausgewertete Anforderungsspezifikationen zur Verfügung.

1.4 Vorgehensweise und Gliederung

In Kapitel 2 wird zunächst ein Überblick über die Grundlagen zu den verschiedenen Softwareentwicklungsmethoden gegeben und die wesentlichen Unterschiede werden herausgearbeitet. Der in agilen und konventionellen Vorgehensmodellen unterschiedliche Umgang mit dem Kunden, mit der Anforderungsanalyse und den Anforderungsänderungen stehen hierbei im Mittelpunkt der Beobachtungen. Darauf aufbauend werden in Kapitel 3 Ausgangshypothesen und erwartete Einflussfaktoren dokumentiert. Besonders interessant hierbei die Klassifizierung der Anforderungen und ihre Auswirkung auf die Ergebnisse.

In Kapitel 4 werden mithilfe der GQM-Methode geeignete Metriken gezeigt und ein Messplan erstellt. Zuerst werden Ziele definiert, die in direktem Zusammenhang mit den Hypothesen stehen. Zu diesen Zielen werden Fragen formuliert, deren Antworten später mit den Hypothesen verglichen werden.

Kapitel 5 beinhaltet die Analyse. Hier werden noch einmal alle Hypothesen aufgegriffen und mit den Ergebnissen aus der Messung verglichen. Kapitel 6 bewertet die Gültigkeit anhand von Einflüssen während der Messung.

Kapitel 7 gibt eine Zusammenfassung der Ergebnisse dieser Arbeit und einen Ausblick auf Aspekte, die in zukünftigen Arbeiten vertieft werden könnten.

2 Grundlagen

Auf der einen Seite gibt es die klassischen Vorgehensmodelle zur Softwareentwicklung. Bekannte Beispiele sind das Wasserfall- und Spiral-Modell. Diesen klassischen Modellen gemein ist die Aufteilung der Softwareentwicklung in eine feste Folge von Tätigkeiten, die je nach Modell ein- oder mehrmals iterativ durchlaufen werden.

Daneben gibt es agile Vorgehensmodelle, bekannt sind SRUM (vgl. [Schw04]) und eXtreme Programming (vgl. [Bec+04]). Bei diesen agilen Methoden liegt der Schwerpunkt weniger auf einer detaillierten Dokumentation der Anforderungen, sondern mehr auf der ständigen Zusammenarbeit zwischen den Entwicklern und dem Kunden.

2.1 Das Prinzip konventioneller Softwareentwicklung

Die Herausforderung bei den klassischen Modellen besteht darin, die Anforderungen über den langen Zeitraum zwischen Anforderungsanalyse und Einführung des Systems beim Kunden und damit bis zum Feedback des Kunden stabil zu halten. Dem Problem tritt man mit einer sehr genauen und umfassenden Durchführung der Tätigkeiten und einer ausführlichen schriftlichen Dokumentation der Ergebnisse entgegen. Ein Feedback des Kunden während des Projekts und eine nachträgliche Änderung der Anforderungen sind nur sehr eingeschränkt möglich. So wird das entstandene Produkt sehr nah an den ursprünglich vereinbarten Anforderungen sein, aber kaum oder keine aktuellen Wünsche des Kunde berücksichtigen.

Diese und andere Schwierigkeiten werden im Folgenden anhand von Modell-Beispielen beleuchtet.

2.1.1 Das Wasserfall-Modell

Das Wasserfallmodell besteht aus einer wohldefinierten Abfolge von Phasen. Die Ergebnisse einer abgeschlossenen Phase sind bindende Vorgabe für die folgende Phase. Ein wiederholtes Durchlaufen der Phasen und somit eine Korrektur von Ergebnissen sind nicht vorgesehen. Alle Anforderungen an die zu entwickelnde Software müssen im Voraus festgelegt werden.

Während der ersten drei Phasen sind keine Prototypen vorgesehen, um die Anforderungsdefinition zu überprüfen. Tests werden erst kurz vor Vollendung des Produkts gemacht. Bei grundlegenden Fehlern oder Änderungswünschen des Kunden kommt es dann zu einem hohen Kostenaufwand, da es durch die fest einzuhaltende Abfolge der Phasen des Wasserfall-Modells meist zu einem Neustart der Entwicklung kommen muss. In der folgenden Abbildung sind die einzelnen Phasen des Wasserfall-Modells dargestellt.

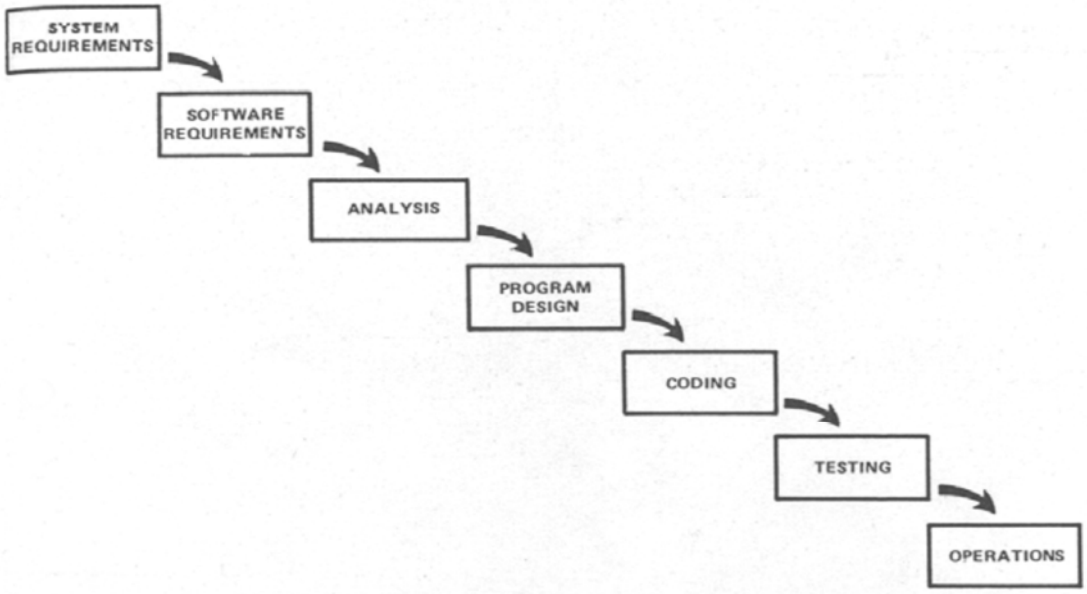

Abbildung 1: Wasserfallmodell [Schn05]

Die Schwierigkeit des Vorgehensmodells besteht also darin, dass davon ausgegangen wird, dass keine Fehler in der Planung gemacht werden und der Kunde an seiner ursprünglichen Anforderungsspezifikation festhält. Da der Kunde ebenfalls Lern- und Entwicklungsprozessen unterliegt, ist diese Annahme sehr unwahrscheinlich. Das Risiko, den Kunden am Ende des Projekts mit dem fertigen Produkt nicht zufriedenstellen zu können, ist hoch.

2.1.2 Das Spiral-Modell

Auch das Spiral-Modell ist in einzelne Phasen unterteilt, deren Ergebnisse der jeweils folgenden Phase als Grundlage dienen. Jedoch werden hier die Phasen iterativ durchlaufen. In jeder Iteration werden alle Phasen durchlaufen und damit auch eventuelle Risiken abgewogen. Die in das Vorgehensmodell integrierte Erstellung von Prototypen bildet eine bessere Grundlage für Diskussionen zwischen den Entwicklern und dem Kunden als abstrakte Dokumente.

Der iterative Durchlauf gibt dem Modell eine gewisse Flexibilität in Bezug auf sich ändernde Anforderungen und stellt einen großen Vorteil dem Wasserfall-Modell gegenüber dar. Ein frühzeitiger Abbruch bei unüberwindbaren Risiken und eine damit verbundene Kosteneinsparung gegenüber dem Wasserfall-Modell sind möglich.

Aber natürlich gibt es neben den positiven Aspekten auch Schwachpunkte des Modells. „*Das Projekt kann nicht in seiner Gesamtheit geplant werden. Die Planungen und Zielsetzungen beziehen sich immer nur auf einen Zyklus*“ [LL06, S.168ff]. Dennoch ist die Reaktionsfähigkeit hinsichtlich auftretender Änderungen eingeschränkt, eine Anpassung ist zeit- und kostenintensiv. Außerdem ist eine Kommunikation zwischen den Entwicklern und dem Kunden zwar vorgesehen, bekommt aber hier nicht den Stellenwert wie bei agilen Methoden eingeräumt.

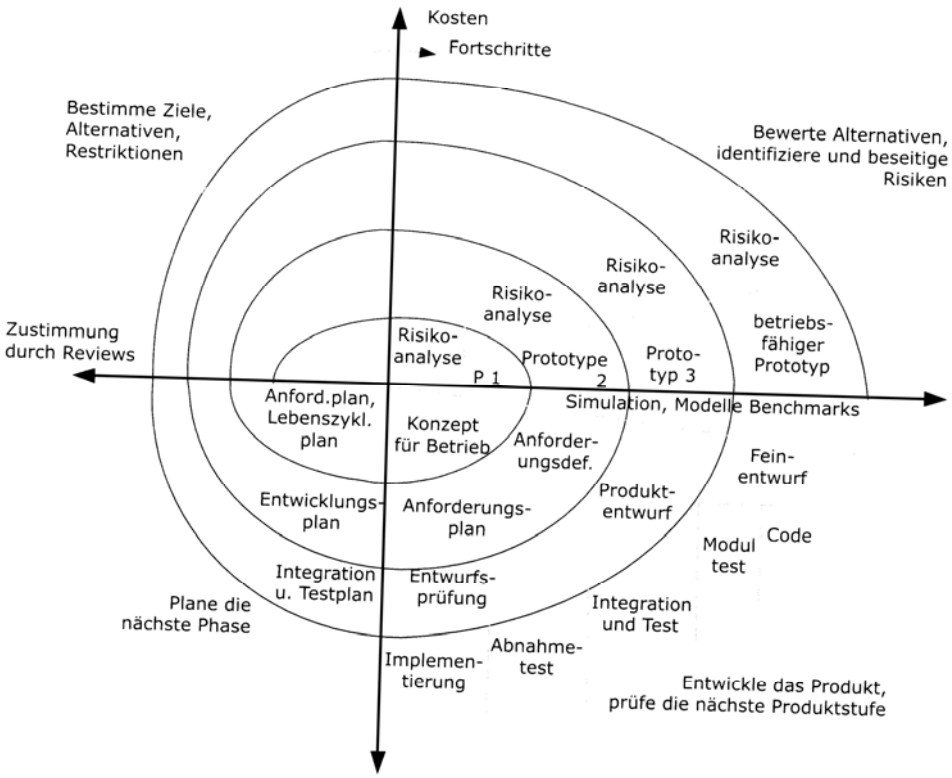

Abbildung 2: „Spiralmodell nach [Boeh88]“ [Schn07]

2.2 Das Prinzip agiler Softwareentwicklung

Agile Methoden zeichnen sich aus durch kooperative Zusammenarbeit zwischen den Entwicklern und dem Kunden, auf eine detaillierte Dokumentation der Anforderungen wird zugunsten höherer Flexibilität verzichtet. Änderungen der Anforderungen sind von Anfang an eingeplant und auch erwünscht. Das wichtigste Anliegen agiler Methoden ist, dem Kunden möglichst schnell ein funktionsfähiges Programm zu präsentieren, das zunächst die wichtigsten vom Kunden gewünschten Funktionen enthält. Auf dieser Basis kann dann mit Entwicklern und Auftraggebern über mögliche Änderungen und Erweiterungen des Produkts diskutiert werden.

Diese Werte werden in dem „*Manifesto for Agile Software Development*“ [Bec+AM] erklärt:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

Zu diesen vier Werten kommen zwölf Prinzipien. An dieser Stelle seinen sechs Prinzipien genannt [Bec+AP]:

- *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*

- *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- *[…]*
- *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- *Working software is the primary measure of progress.*
- *[…]*
- *Simplicity--the art of maximizing the amount of work not done--is essential.*
- *[…]*

Anhand dieser Prinzipien ist zu erkennen, dass agile Methoden nicht einen strengen Plan verfolgen, der zu Beginn des Projekts erstellt wurde, sondern dass spätere Änderungen ohne großen zusätzlichen Aufwand aufgenommen werden können. Späte Änderungen werden sogar als Chance für eine bessere Software gesehen. Es wird mehr Wert auf die Praxis, das Produkt und die Menschen, die an einem Projekt arbeiten, gelegt, als auf aufwändige Dokumentationen, welche während des Projektes ohnehin veralten und damit nicht brauchbar sind. Freigewordene Ressourcen durch den Verzicht auf oft redundante Dokumente sollen für die Entwicklung funktionierender Software eingesetzt werden.

2.2.1 SCRUM

SCRUM nimmt an, dass der Prozess der Softwareentwicklung so komplex ist, dass er sich im Voraus nicht komplett planen lässt. SCRUM zielt eher darauf ab, mittelfristig zu planen und eine Möglichkeit für schnelle Reaktionen zu schaffen.

SCRUM funktioniert nach dem folgenden Prinzip: Die Software-Entwicklung wird in Iterationen unterteilt, den sogenannten Sprints, die etwa 30 Tage dauern und auf ein vorher festgelegtes Ziel hinarbeiten. Die verhältnismäßig kurze Iterationszeit schafft schnelle Reaktionszeiten. Die Anforderungen werden als noch zu entwickelnde Funktionalitäten zu Beginn jedes Sprints in einer priorisierten Liste, dem Product Backlog, festgehalten. Diese Liste ist grundsätzlich allen Mitarbeitenden zugänglich, wird aber von einem Product Owner geführt. Während des Sprints bleiben die Anforderungen fixiert. Zu Beginn jeden Tages wird ein SCRM-Meeting gehalten, um die täglichen Entwicklungen zu reflektieren. Am Ende jedes Sprints wird das Produkt in einem Sprint Review Meeting informell bewertet. Hier bietet sich dann die Gelegenheit zum Informationsaustausch zwischen den Entwicklern und dem Kunden.

Problematisch ist, dass es bei SCRUM keine Vorgaben für die Erstellung der Anforderungen gibt. Natürlich lässt sich SCRUM mit anderen agilen Entwicklungsmethoden kombinieren, aber es liegt am Product Owner eine passende Vorgehensweise zu finden. Außerdem bieten die Anforderungen keine Grundlage für einen Vertrag, das Endprodukt ist nicht spezifiziert.

2.2.2 eXtreme Programming (XP)

Die Grundlage dieser Studienarbeit bildet eine Datenerhebung in einem XP-Labor in studentischem Umfeld. Ein XP-Labor ist eine Intensivübung für Studenten, um Erfahrungen mit XP-Praktiken sammeln zu können (siehe auch Abschnitt 4.5.1).

XP legt eine Menge von flexiblen Aktivitäten fest, über deren Abfolge und Intensivität die Entwickler und Projektleiter den aktuellen Gegebenheiten entsprechend entscheiden können. Ziel ist es, möglichst schnell hochwertigen Code zu erstellen, der die Anforderungen des Kunden erfüllt.

Wie in allen Methoden spielen die Anforderungen auch bei XP eine zentrale Rolle, da sie die Wünsche des Kunden an das System darstellen und die Kundenzufriedenheit das höchste Ziel ist. Die Anforderungsanalyse in XP ist aber absichtlich eher skizzenhaft. Unter in XP gegebenen Umständen kann „*man die Entwicklung mit einem einfachen Plan beginnen und diesen Plan im weiteren Verlauf fortwährend weiter ausarbeiten*“ [Bec00, S.64]. Die Anforderungen werden vom Kunden selbst in sogenannte Story Cards geschrieben und sind möglichst kurz und für Entwickler und Kunden gleichermaßen verständlich zu halten. Zu beschreiben sind wichtige Bestandteile, die der Kunde vom System erwartet. „*A user story is nothing more than an agreement that the customer and developers will talk together about a feature*” [Bec+00, S.46].

Die in der ersten Iteration erstellten Story Cards enthalten die für den Kunden wichtigsten Anforderungen an das System. Auf dieser Basis werden während des Projekts laufend Stories hinzugefügt, bestehende geändert oder sogar entfernt. Aus der entstandenen Menge wird für jede Iteration ein Teil herausgesucht, der dann während dieser Iteration entwickelt wird.
Änderungen der Anforderungen sind bei XP von vorn herein einkalkuliert und akzeptiert. „*Jedes Problem wird durch eine Reihe kleinerer, jedoch wirkungsvoller Änderungen gelöst*“ [Bec00, S.38].

Die Implementierung findet dann in Paaren statt (Pair Programming). Jedes Paar teilt sich einen Computer, unterstützt sich bei der Implementierung, achtet auf Fehler und macht gegenseitig Verbesserungsvorschläge. Nach jeder Änderung im Code werden Tests durchgeführt. So wird einerseits die Arbeit der Entwickler überprüft und andererseits dem Kunden gezeigt, dass er das bekommt, was er in Auftrag gegeben hat.

Der Kunde ist bei XP das ganze Projekt über vor Ort oder zumindest kurzfristig erreichbar. Er muss außerdem kompetent genug sein, die Anforderungen aller Interessensgruppen zusammenzuführen.

Die folgenden Abbildungen zeigen eine Übersicht über die XP-Praktiken und den iterativen Ablauf eines XP Projekts:

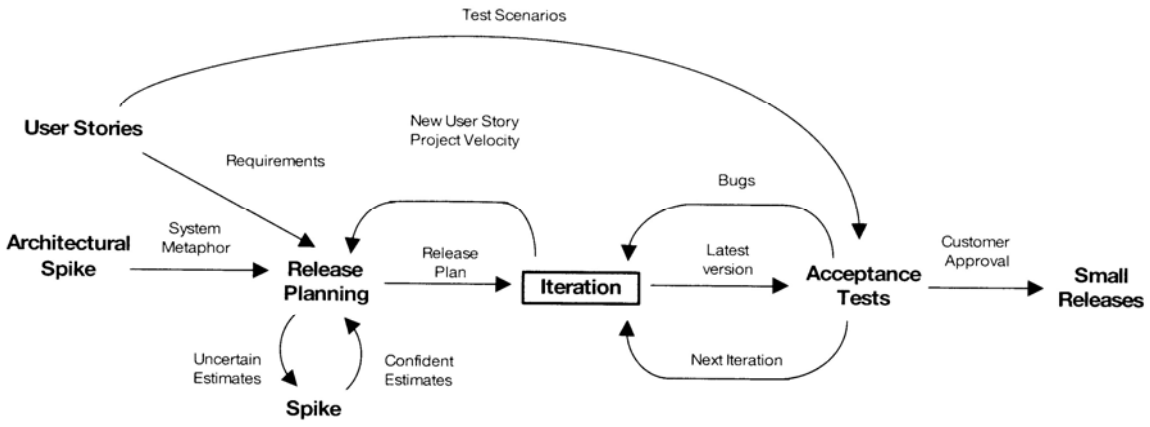

Abbildung 3: Ein XP-Projekt [ZGK04, S.102]

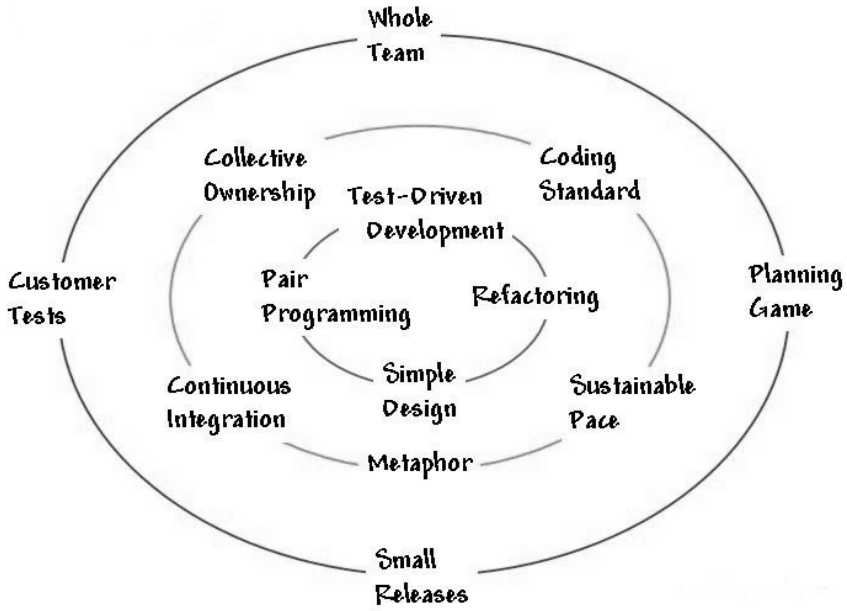

Abbildung 4: XP-Practices [Jef01]

Die Methode XP besticht also durch folgende Grundprinzipien [Bec00, S.37]:

- *Unmittelbares Feedback*
- *Einfachheit anstreben*
- *Inkrementelle Veränderung*
- *Veränderungen wollen*
- *Qualitätsarbeit*

„*XP eignet sich* [aber] *nicht für jedes Softwareprojekt, sondern wurde besonders für Software entwickelt, deren Anforderungen sich in regelmäßigen Abständen ändern, oder für Softwareprojekte, in denen die genauen Anforderungen vom Kunden erst im späteren Verlauf festgelegt werden*“ [ZGK04, S.102].

Unmöglich ist es z.B. auch bei XP, die viel zu wenig detaillierten Anforderungen als Vertragsgrundlage zu nutzen. Genauso schwierig ist die Umsetzung des Kunden-vor-Ort. Die ständige Präsenz erfordert einen großen Aufwand für den Kunden, auch wenn die Inanspruchnahme seitens der Entwickler vielleicht gar nicht so groß ist. Der Kunde muss jederzeit ansprechbar sein. Die gesprächsfreie Zeit für andere Arbeiten zu nutzen, fällt wegen der häufigen Unterbrechungen schwer. Das Projekt fordert scheinbar die ganze Aufmerksamkeit des Kunden. Das wurde auch in den Beobachtungen des XP-Labors zu dieser Studienarbeit deutlich.

2.3 Vergleich der verschiedenen Prinzipien

Viele späte Anforderungsänderungen in einem Projekt, schnellere Releasezyklen und Marktdruck machen es heute schwierig mit konventionellen Entwicklungsmethoden brauchbare Software zu entwickeln. Vorgehenspläne haben nicht genug Bestand. Aufwändige Dokumentationen sind schnell nicht mehr aktuell. Verbesserung verspricht die agile Softwareentwicklung, Anforderungen nehmen hier eine andere Rolle ein.

Agile Methoden bieten eine gute Einbindung des Kunden. Bei XP ist das z.B. der sogenannte On-Site-Customer. Die Kommunikation findet also zwischen Menschen statt. Bei konventionellen Entwicklungsmethoden dagegen ist der Kontakt mit dem Kunden zwischen der Anforderungsspezifikation und der Abnahme des Produkts sehr unwahrscheinlich. Die Kommunikation findet hier über Dokumente statt.

So vergeht viel Zeit, bis der Kunde den ersten Prototyp zu sehen bekommt und ein Feedback geben kann. Bei den agilen Methoden wird dagegen schnell brauchbare Software geliefert, die der Kunde vor Ort schnell bewerten kann. Kleine Iterationen ermöglichen, die Anforderungen weiterzuentwickeln und das Resultat zu optimieren.

Die Anforderungen werden bei einer agilen Herangehensweise in der Anforderungsanalyse nur grob skizziert, um keine Zeit für aufwendige Dokumentationen zu verlieren. Anders aber bei konventionellen Methoden, bei denen viel Wert auf eine möglichst ausführliche Anforderungsspezifikation und Dokumentation gelegt wird.

Änderungen der Anforderungen werden bei agilen Methoden erwartet und akzeptiert, man möchte flexibel bleiben. Bei der klassischen Softwareentwicklung plant man lieber voraus, Änderungen sind nicht erwünscht.

Natürlich haben klassische Entwicklungsmethoden auch ihre Vorteile. So kann die Anforderungsspezifikation als Vertragsgrundlage dienen. Der Kunde hat durch den festen Vorgehensplan eine genaue Vorstellung davon, was die Entwickler gerade oder als nächstes tun. Es ist schwer, einen Kunden ohne genauen Vorgehensplan von sich zu überzeugen. Außerdem erfordern einige Projekte ein hohes Maß an Sicherheit, dafür bieten konventionelle Entwicklungsmethoden die bessere Grundlage.

Zu dem Vergleich zwischen agilen und konventionellen Entwicklungsmethoden lässt sich aus der Distanz sagen, dass die Ansätze sich zwar in Ihrer Flexibilität, in der Rolle des Kunden und dem Umgang mit Anforderungsänderungen unterscheiden, ihr letztliches Ziel aber dennoch das gleiche ist – die Entwicklung eines qualitativ hochwertigen Softwareprodukts. Gerade diesen groben Vergleich gilt es in den folgenden Kapiteln durch empirische Fakten im Detail zu führen.

2.4 Arten von Anforderungen

Es gibt verschiedene Ansätze zur Klassifikation von Anforderungen. Am häufigsten findet man die Unterteilung in funktionale und nicht-funktionale Anforderungen.

Eine Anforderung ist *„eine Bedingung oder eine Fähigkeit funktionaler oder nicht-funktionaler Natur, welche ein Produkt erfüllen bzw. haben muss*“ [ZGK04, S.223].

2.4.1 Funktionale Anforderungen

Anforderungen, die *„vom System selbstständig ausgeführt werden sollen, Interaktionen des Systems (Eingaben, Ausgaben) mit menschlichen Nutzern und Anforderungen zu allgemeinen, funktionalen Vereinbarungen und Einschränkungen [Rup06]“* [Bou08, S.11] sind, sind funktional. Diese Anforderungen legen also fest, wie sich die Software verhalten soll.

2.4.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen *„sollen alle jene Merkmale der Software darstellen, die zum Gelingen der Interaktion zwischen Anwender und System maßgeblich beitragen, aber mit der direkten Eingabe-Verarbeitung-Ausgabe-Kette von Daten nichts zu tun haben*“ [ZGK04, S.225]. Sie sagen etwas darüber aus, welche Eigenschaften die Software haben soll. Nicht-funktionale Anforderungen lassen sich in verschiedene Typen unterteilen. Einige gängige Typen sind Benutzbarkeit, Zuverlässigkeit, Effizienz und Wartbarkeit. Aber auch technische Anforderungen gehören zu den nicht-funktionalen Anforderungen.

2.4.3 Granularität von Anforderungen

Sowohl funktionale als auch nicht-funktionale Anforderungen lassen sich außerdem in Bezug auf ihre Granularität klassifizieren. So können vom Kunden entweder sehr grobe oder sehr detaillierte Anforderungen gestellt werden. So bietet sich als indirektes Maß für den Grad der Granularität bei funktionalen Anforderungen eine Einteilung in Geschäftsziele, Benutzerziele und Subfunktionen an. Bei nicht-funktionalen Anforderungen kann man von abstrakten, konkreten und messbaren Anforderungen sprechen.

2.4.4 Hinweis zur Klassifikation

Bei der in Kapitel 5 folgenden Datenerhebung hat sich die Einordnung in funktionale oder nicht-funktionale Anforderung als nicht immer ganz eindeutig erwiesen. Einige Anforderungen enthielten gleichermaßen funktionale und nicht-funktionale Aspekte. Wie die Anforderungen für die Datenerhebung dennoch eindeutig einer Art von Anforderungen zugewiesen werden können, wird in Kapitel 3 noch einmal aufgegriffen und anhand von Beispielen erläutert.

3 Hypothesen und erwartete Einflüsse

Auf der Basis der erarbeiteten Unterschiede zwischen agilen und konventionellen Entwicklungsmethoden können folgende Annahmen zu den Ergebnissen der Datenerhebung im XP-Labor gemacht werden, die es dann in den folgenden Kapiteln zu überprüfen und zu bewerten gilt.

Durch die bessere Kommunikation zwischen den Entwicklern und dem Kunden lässt sich vermuten, dass in vielen kleinen Gesprächen bei XP mehr über nicht-funktionale als über funktionale Anforderungen gesprochen wird als es bei einer konventionellen Entwicklungsmethode der Fall wäre. Denn durch kurze Iterationen und die frühe Lieferung brauchbarer Software, bekommt der Kunde genauere Vorstellungen über das gewünschte Endprodukt. Seine Anforderungen werden im Laufe des Projekts präziser und zielen mehr auf die Eigenschaften der Software ab.

Dem Kunden werden beim Testen der Software besonders die Eigenschaften auffallen, die die Benutzbarkeit (Usability) betreffen. Dadurch kann man annehmen, dass die meisten der nicht-funktionalen Anforderungen die Benutzbarkeit (Usability) betreffen. Der Anteil an Usability-Anforderungen an allen nicht-funktionalen Anforderungen in einem XP Labor wird vermutlich größer sein als bei einer klassischen Entwicklungsmethode.

Durch die skizzenhafte Anforderungsanalyse innerhalb eines XP-Labors ist davon auszugehen, dass in der ersten Iteration nur die wenigsten aller bis zum Ende des Projektes gestellten Anforderungen ermittelt werden. Viele werden nach Gesprächen mit dem Kunden entweder geändert oder verworfen.

Durch die Kombination von On-Site-Customer und kleinen Iterationen wird bei XP immer wieder über schon zuvor erhobene Anforderungen gesprochen. Mit jedem weiteren Gespräch wird der Grad der Granularität feiner werden. Insgesamt wird in XP vermutlich mehr über Subfunktionen und messbare Anforderungen gesprochen als über Geschäftsziele und abstrakte Anforderungen.

3.1 Hypothesen in der Übersicht

1. Hypothesen den Vergleich zwischen agilen und konventionellen Entwicklungsmethoden betreffend:

    1.1. Die Anzahl der nicht-funktionalen Anforderungen ist im Verhältnis zu den funktionalen Anforderungen bei XP höher als bei konventioneller Softwareentwicklung.
    1.2. Die Anzahl der Usability-Anforderungen ist im Verhältnis zu anderen nicht-funktionalen Anforderungen bei XP höher als bei konventioneller Softwareentwicklung.

2. Hypothesen XP betreffend:

    2.1. Die Usability-Anforderungen machen bei XP den größten Teil der nicht-funktionalen Anforderungen aus.

2.2. Die meisten Anforderungen entstehen bei XP erst im Laufe des Projektes.
2.3. Die Basis-Anforderungen bleiben meist nicht in ihrer ursprünglichen Version erhalten. Sie werden entweder während des Projekts geändert oder verworfen.
2.4. In XP wird mehr über Subfunktionen und messbare Anforderungen gesprochen als über Geschäftsziele und abstrakte Anforderungen.

3.2 Mögliche Einflüsse

In einem zweistündigen Planspiel (Agil Hour, vgl. [LS05]) wurde vor der eigentlichen Datenerhebung eine Probemessung durchgeführt. Die in dem Planspiel erhobenen Anforderungen wurden protokolliert und klassifiziert. Hier hat sich gezeigt, dass die Klassifikation der Anforderungen nicht immer eindeutig ist. Oft enthielt eine Anforderung sowohl einen funktionalen als auch einen nicht-funktionalen Aspekt.

Die Frage ist, ob es eine Möglichkeit gibt zu entscheiden, welcher Aspekt mehr wiegt, oder ob es tatsächlich unentscheidbare Fälle gibt. Dazu werden im Folgenden einige Beispiele gegeben.

1. Beispiele für funktionale Anforderungen:

   „Pro Use Case soll ein Fragebogen erstellt werden können.“ (Funktion: Text erstellen)

   „Der Fragebogen muss gespeichert werden können.“ (Funktion: Text speichern)

   „Die Auswertung der Fragebögen soll zeigen, wie groß die Wahrscheinlichkeit ist, dass ein bestimmtes Risiko eintritt.“ (Funktion: Berechnung)

2. Beispiele für nicht-funktionale Anforderungen:

   „Das Risiko-Management soll auch anderen zur Verfügung gestellt werden können.“

   Hier geht es nicht darum, was das Produkt tun soll, sondern um die Eigenschaft der Wiederverwendbarkeit des Produkts.

   „Über eine Schnittstelle soll sichergestellt werden, dass nur der Projektleiter Zugriff hat.“

   Der Aspekt der Sicherheit ist ausschlaggebend für die Klassifizierung.

   Nur Use Cases, die aktuell im Projekt sind, sollen berücksichtigt werden.“

   Hier geht es nicht mehr um die Berechnung an sich, ein Ergebnis würde ja auch mit nicht aktuellen Use Cases erzielt. Der Aspekt der Korrektheit spielt hier die wesentliche Rolle.

3. Manchmal zeigt sich, dass ein Satz mit einer Anforderung, die scheinbar sowohl dem funktionalen als auch dem nicht-funktionalen Typ zuzuordnen ist, teilbar ist in zwei Sätze und somit auch in zwei Anforderungen, die dann getrennt voneinander klassifiziert werden können.

   „Use Cases sollen für den Projektleiter sichtbar ID und Titel haben.“

Einerseits geht es um die Funktion, dass jedem Use Case ein Titel und eine ID zugewiesen werden, andererseits sollen dem Projektleiter diese Informationen zur besseren Benutzbarkeit (Usability) angezeigt werden.

Den Satz kann man nun folgendermaßen aufteilen:

a. „Use Cases sollen Titel und ID haben."

b. „Titel und ID der Use Cases sollen für den Projektleiter sichtbar sein."

Die Klassifikation wird so klarer. In Teil a. geht es um den funktionalen, in Teil b. um den nicht-funktionalen Aspekt.

In der späteren Klassifikation sollten die Anforderungen also so fein wie möglich notiert und betrachtet werden.

4. Anders bei dem folgenden Beispiel:

„Der Report soll sich die Dateien vom Server automatisch holen, so dass kein manuelles Eingreifen mehr nötig ist."

Funktional: Report erstellen.
Nicht-funktional: Manuelle Ausführung soll verhindert werden, Aspekt Usability.

Diese Anforderung könnte man sowohl funktional als auch nicht-funktional (Usability) einstufen, aber eine Unterteilung des Satzes ist nicht mehr möglich. Die einzige Möglichkeit, eine Entscheidung treffen zu können, ist hier, den Kontext (das Protokoll) genauer zu betrachten, um zu sehen, aus welchem Grund der Kunde die Anforderung gestellt hat.

Genauso bei dem folgenden Beispiel:

„Die Umfrage soll mit einer neuen Menge von Use Cases neu gestartet werden können."

Funktional: Neustart möglich.
Nicht-funktional: Aspekt Robustheit.

Die Klassifizierung aus dem Kontext heraus scheint erstmal subjektiv und unsicher. Allerdings sollte eine Quote unsicherer Klassifizierungen bis rund 10% das Ergebnis nicht wesentlich beeinflussen und toleriert werden. In Kapitel 6.1 sind die Ergebnisse dieser Studienarbeit diesbezüglich untersucht worden.

4 Erstellung des Messplans

4.1 Die GQM-Methode

GQM steht für Goal-Question-Metric. Diese Methode dient dazu, systematisch aus Zielen geeignete Metriken zu bestimmen. „*Die Grundidee von GQM ist […]: Man soll nicht das messen, was leicht zu messen ist, sondern das, was man braucht, um seine Verbesserungsziele zu erreichen*“ [Schn07, S.69].

Dazu definiert man zuerst die eigenen Ziele (Goal), zu denen man dann „*Fragen* [(Question)] *formuliert und Metriken* [(Metric)] *sucht oder definiert, mit denen man die Fragen beantworten kann*“ [Schn07, S.69]. Aus dem entstandenen Ziel- bzw. GQM-Baum wird ein Messplan entwickelt, der genau vorgibt, wer wie welche Messungen durchführt. Darauf folgt die eigentliche Messung, nach der man dann die Ergebnisse „rückwärts einsetzt und so von den Metriken bis zu den Zielen verfolgt“ [Schn07, S.70]. Das Resultat sind Antworten auf die vor der Messung gestellten Fragen.

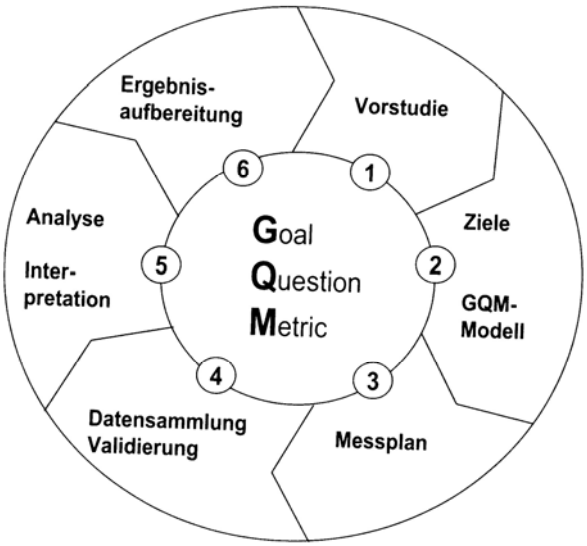

Abbildung 5: Basilis Goal-Question-Paradigm zur iterative Verbesserung [Schn07, S.70]

4.2 Ziele und Facettenbeschreibung

Das Hauptziel ist, die Menge von Anforderungen in XP quantitativ und qualitativ genauer zu untersuchen. Teilweise sollen die Ergebnisse mit den vorliegenden Daten aus den Softwareprojekten (SWP) verglichen werden. Basierend auf den oben genannten Hypothesen zu XP und konventionellen Entwicklungsmethoden ergeben sich die im Zielbaum (siehe Abbildung 6) dargestellten Teilziele.

In der an den Zielbaum anschließenden Tabelle 1 sind die Zielfacetten noch einmal als strukturierte Sätze aufgeschrieben. Im nächsten Kapitel sollen zu den Zielen Fragen gestellt werden. „*Die Hypothesen sind als erwartete Antworten auf die formulierten Fragen aufzufassen [SB99, S.53-58]*“ [Bou08]. Um die Zuordnung später zu erleichten, sind in der Tabelle die Hypothesen schon den Zielen zugeordnet.

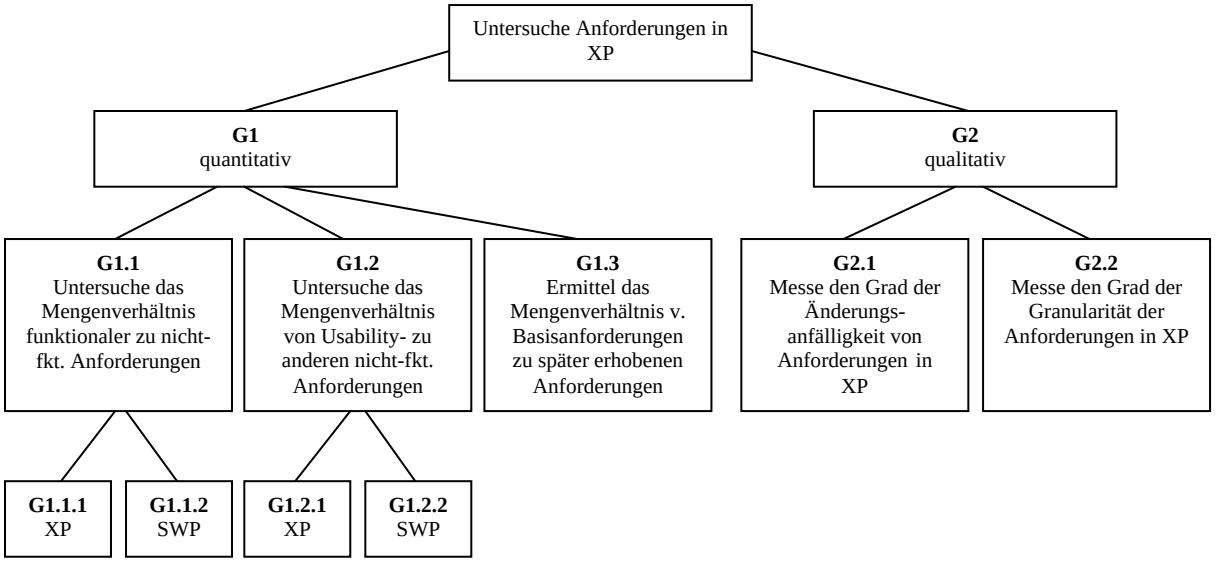

Abbildung 6: Zielbaum

„Eine Facette ist ein Aspekt, nach dem ein Ziel klassifiziert werden kann. In GQM haben sich vier Facetten bewährt, um zu besonders treffenden Fragen und Metriken zu gelangen“ [Schn07, S.73]. Diese vier Facetten sind Zweck, Qualitätsaspekt, Betrachtungsgegenstand und Perspektive. Eine Möglichkeit ist die Facetten als strukturierte Sätze aufzuschreiben.

Für diese Messung wurden die Facetten Zweck, Qualitätsaspekt und Betrachtungsgegenstand gewählt. Da die Messung auf konkrete Zahlen als Ergebnis abzielt, spielt die Perspektive keine entscheidende Rolle. Die Zahlen haben unter Berücksichtigung der Bewertungskriterien (siehe Kapitel 6) Gültigkeit, ob sie nun wie in dieser Arbeit aus der Sicht der Forschung seitens des Fachgebiets Software Engineering oder aber aus der Sicht eines Projektteilnehmers betrachtet werden.

Beispiel für die Darstellung der Facetten als strukturierter Satz:
Untersuche (Zweck) *das Verhältnis* (Qualitätsaspekt) f*unktionaler zu nicht-funktionaler Anforderungen in XP* (Betrachtungsgegenstand).

So kommen oben beschriebene Ziele in die folgende Form:

Hypothese	Ziel	Facetten
1.1.	G1.1.1	Untersuche das Verhältnis funktionaler zu nicht-funktionaler Anforderungen in XP.
1.1.	G1.1.2	Untersuche das Verhältnis funktionaler zu nicht-funktionaler Anforderungen in Softwareprojekten.
1.2. 2.1.	G1.2.1	Untersuche das Verhältnis von Usability- zu anderen nicht-funktionalen Anforderungen in XP.
1.2.	G1.2.2	Untersuche das Verhältnis von Usability- zu anderen nicht-funktionalen Anforderungen in Softwareprojekten.
2.2.	G1.3	Ermittel das Verhältnis von Basisanforderungen zu später erhobenen Anforderungen.
2.3.	G2.1	Messe den Grad der Änderungsanfälligkeit von Anforderungen in XP.
2.4.	G2.2	Messe den Grad der Granularität der Anforderungen in XP.

Tabelle 1: Zielfacetten

4.3 Fragen

Ziel	Frage
G1.1.1	Wieviele der Anforderungen in XP sind funktional? Wieviele der Anforderungen in XP sind nicht-funktional?
G1.1.2	Wieviele der Anforderungen in Softwareprojekten sind funktional? Wieviele der Anforderungen in Softwareprojekten sind nicht-funktional?
G1.2.1	Wieviele der nicht-funktionalen Anforderungen in XP sind Usability-Anforderungen? Wieviele der Anforderungen in XP sind andere nicht-funktionale Anforderungen?
G1.2.2	Wieviele der nicht-funktionalen Anforderungen in Softwareprojekten sind Usability-Anforderungen? Wieviele der Anforderungen in Softwareprojekten sind andere nicht-funktionale Anforderungen?
G1.3	Wieviele Anforderungen in XP wurden im 1. Planning Game erhoben (Basisanforderungen)? Wieviele Anforderungen kommen in späteren Iterationen noch dazu?
G2.1	Wieviele der Basisanforderungen werden in späteren Iterationen wieder aufgegriffen (geändert oder gelöscht)? Wieviele Basisanforderungen bleiben in ihrer ursprünglichen Form erhalten?
G2.2	Wieviele Anforderungen sind abstrakt bzw. handeln von Geschäftszielen? Wieviele Anforderungen sind konkret bzw. handeln von Benutzerzielen? Wieviele Anforderungen sind messbar bzw. handeln von Subfunktionen?

Tabelle 2: Fragen nach GQM

4.4 Metriken

Ziel	Metriken
G1.1.1	Erhebe Menge der funktionalen Anforderungen im XP Labor. Gesamtanzahl – Anz. fkt. Anforderungen = Anz. nicht-fkt. Anforderungen
G1.1.2	Erhebe die durchschnittliche Anzahl funktionaler Anforderungen in Softwareprojekten. Gesamtanzahl – Anz. fkt. Anforderungen = Anz. nicht-fkt. Anforderungen
G1.2.1	Erhebe Menge der Usability-Anforderungen im XP Labor. Menge der nicht-funktionalen Anforderungen siehe Metrik G1.1.1. Nicht-fkt. Anf. – Usability-Anf. = andere nicht-fkt. Anf.
G1.2.2	Erhebe die durchschnittliche Anzahl Usability-Anforderungen in Softwareprojekten. Menge der nicht-funktionalen Anforderungen siehe Metrik G1.1.2. Nicht-fkt. Anf. – Usability-Anf = andere nicht-fkt. Anf
G1.3	Erhebe Menge von Basisanforderungen nach dem 1. Planning Game. Erhebe Menge von Anforderungen nach dem 2.-4. Planning Game. Bilde Summe der Anforderungen aus dem 2.-4. Planning Game.
G2.1	Erhebe Menge von Basisanforderungen pro Story Card. Erhebe Menge der Änderungen pro Story Card.
G2.2	Erhebe Menge von abstrakten Anforderungen bzw. Geschäftszielen. Erhebe Menge von konkreten Anforderungen bzw. Benutzerzielen. Erhebe Menge von messbaren Anforderungen bzw. Subfunktionen.

Tabelle 3: Metriken nach GQM

4.5 Messplan

4.5.1 Messplan für XP

Die Durchführung der Messung von XP ist bei zwei parallel laufenden XP-Labor-Projekten möglich gewesen. Jedem Projekt sind acht Studenten zugeteilt. Ein Projekt beschäftigte sich mit dem Thema „Ressourcenplanungs-Tool“ (P1), das andere mit dem Thema „Risikomanagement Webservice“ (P2). Das Ressourcenplanungs-Tool soll im Fachgebiet Software Engineering eingesetzt werden, um Ressourcen terminlich einzuplanen. Der Webservice für das Risikomanagement soll in Softwareprojekten Anwendung finden.

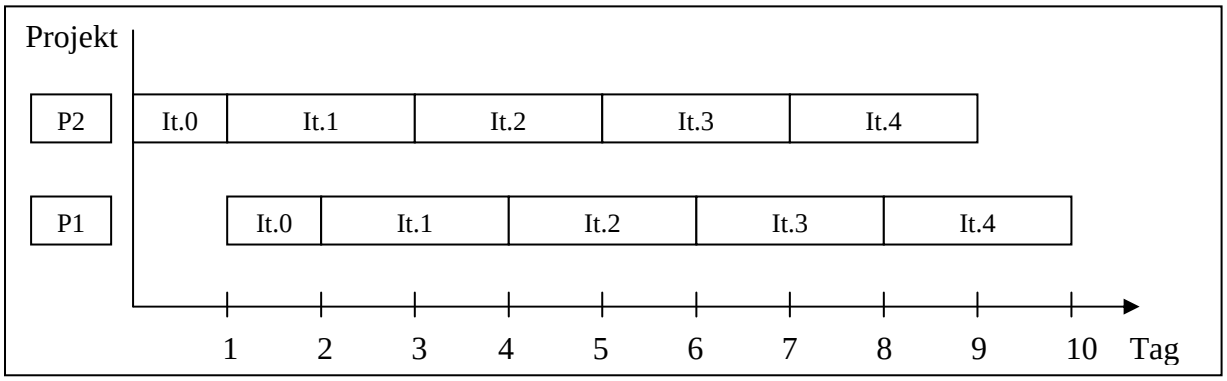

Abbildung 7: Zeitlicher Verlauf der XP-Projekte

Es standen zwei Protokollantinnen zur Verfügung. Im Regelfall waren beide Protokollantinnen anwesend und je einem Projekt zugeordnet. Wenn zu irgendeinem Zeitpunkt insgesamt nur eine Protokollantin zur Verfügung stand, aber in beiden Themengruppen gleichzeitig über Anforderungen gesprochen wurde, so ist dem Projekt „Ressourcenplanungs-Tool“ die höhere Priorität zugewiesen worden.

Um die Wiederholbarkeit der Datenerhebung abschätzen zu können, wurde ein Projekt teilweise doppelt erfasst. Mithilfe der dabei erhobenen Daten wurde festgestellt, dass unterschiedlich Erfasser nur gering Einfluss auf die Messung nehmen (siehe Abschnitt 6.4).

Während der Kundengespräche wurde Gesprochenes zunächst vollständig protokolliert und das Protokoll mit Datum, Uhrzeit, Iterationsnummer und gegebenenfalls mit der Information versehen, ob gerade ein Planning Game oder eine Story-Card-Abnahme stattfindet. Später wertete die Protokollantin die Protokolle aus und füllte zu jeder Anforderung folgendes Klassifizierungs-Formular aus:

Anforderung Nr. _____ Kunde: _____
Gesprächsprotokoll Nr. _____ Erfasser: _____
Anfangszeit: _____
Endzeit: _____
Iteration Nr. _____ Planning Game: ☐ Story-Card-Abnahme: ☐
Zu Story-Card Nr. _____

_____

_____

_____

_____

funktional	☐	Geschäftsziel	☐	Benutzerziel	☐	Subfunktion	☐
Usability	☐	abstrakt	☐	konkret	☐	messbar	☐
andere nicht-funktionale	☐					bestätigt	☐

Abbildung 8: Klassifizierungs-Formular

Die Protokollantin überträgt zunächst die Rahmenbedingungen (Nummer, Uhrzeit, Iteration usw.) vom Protokoll in das Formular. *Hinweis: Die Zeit vor dem ersten Planning Game wird als „0. Iteration“ bezeichnet.* Dann wird die Anforderung in ein bis zwei Sätzen notiert und klassifiziert.

Die Anforderungen können dann den Metriken entsprechend ausgezählt und bewertet, also den Hypothesen zugeordnet werden.

4.5.2 Messplan für Softwareprojekte

Für die Datenerhebung stehen 40 Softwareprojekte zur Verfügung. An diesen Projekten haben Studenten des 5. Semesters teilgenommen. Sie haben in kleinen Gruppen (in der Regel fünf Studierende pro Gruppe) verschiedene Softwareprodukte entwickelt. 16 dieser Projekte wurden in [Bou08] bereits ausgewertet und bilden die Grundlage für den Vergleich zwischen den Softwareprojekten und XP.

5 Analyse

5.1 Vorgehensweise

Für die Analyse ist folgendes Vorgehen vorgesehen:

1. Klassifizierung der Anforderungen

Klassifiziert werden zunächst alle Anforderungen, die aus den Protokollen hervorgehen. Daraus gehen drei Typen von Anforderungen hervor:

- Anforderungen aus Story-,Task- und Bug-Cards (feste Anforderungen),
- flüssige Anforderungen,
- redundante Anforderungen.

Die Anforderungen sollen so fein wie möglich in die Klassifizierungsformulare übertragen werden. Beinhaltet ein Satz zum Beispiel mehrere Anforderungen, so muss der Satz aufgeteilt und jede Anforderung einzeln in ein Formular eingetragen werden.

2. Anwenden der Metriken mithilfe einer Excel-Tabelle (siehe beigelegte CD-ROM)

Die klassifizierten Anforderungen werden in eine Excel-Tabelle übertragen. Redundante Anforderungen müssen hier identifiziert und markiert werden. Sie werden bei der Anwendung der Metriken nicht berücksichtigt.

Die Anwendung der Metriken muss folgende Ergebnisse liefern:

- Anzahl der funktionalen Anforderungen pro Iteration und total
- Anzahl der Usability-Anforderungen pro Iteration und total
- Anzahl der anderen Anforderungen pro Iteration und total
- Anzahl aller nicht-funktionalen Anforderungen pro Iteration und total
- Anzahl aller Anforderungen pro Iteration und total
- Anzahl der abstrakten Anforderungen bzw. der Geschäftsziele pro Iteration und total
- Anzahl der konkreten Anforderungen bzw. der Benutzerziele pro Iteration und total
- Anzahl der messbaren Anforderungen bzw. der Subfunktionen pro Iteration und total
- Anzahl der Anforderungen pro Story Card
- Anzahl der Änderungen jeder Story Card

3. Darstellung der Ergebnisse in Diagrammen

Die Ergebnisse werden in Diagrammen dargestellt. Diese veranschaulichen die Ergebnisse, die wiederum die Antworten auf die gestellten Fragen liefern können. Die Antworten werden dann mit den Hypothesen verglichen und die Hypothesen entweder bestätigt oder widerlegt.

5.2 Messergebnisse

- G1.1.1:
  Untersuche das Verhältnis funktionaler zu nicht-funktionaler Anforderungen in XP

  In beiden beobachteten XP-Projekten wurden in der 0. Iteration mehr funktionale als nicht-funktionale Anforderungen erhoben. Dann aber wurden ebenfalls in beiden Projekten - mit einer einzigen Ausnahme (P2, Iteration 4) - in Iteration 1-4 mehr nicht-funktionale als funktionale Anforderungen erhoben.
  Der Anteil nicht-funktionaler Anforderungen über den gesamten Projektzeitraum gesehen beträgt in P1 57% und in P2 48%.

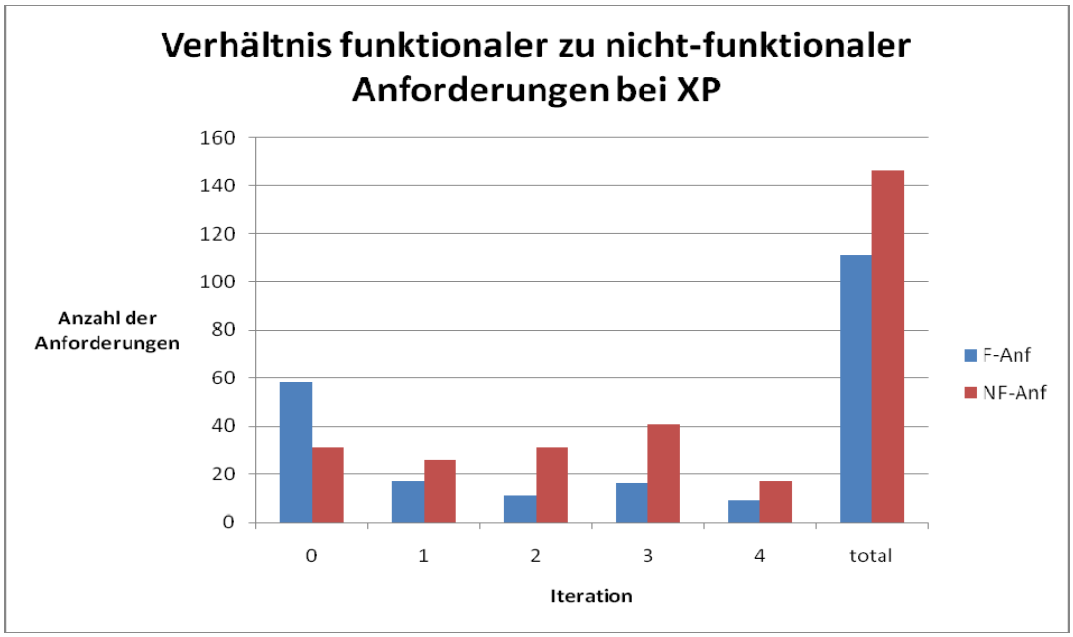

Abbildung 9: Verhältnis funktionaler zu nicht-funktionaler Anforderungen bei XP in P1

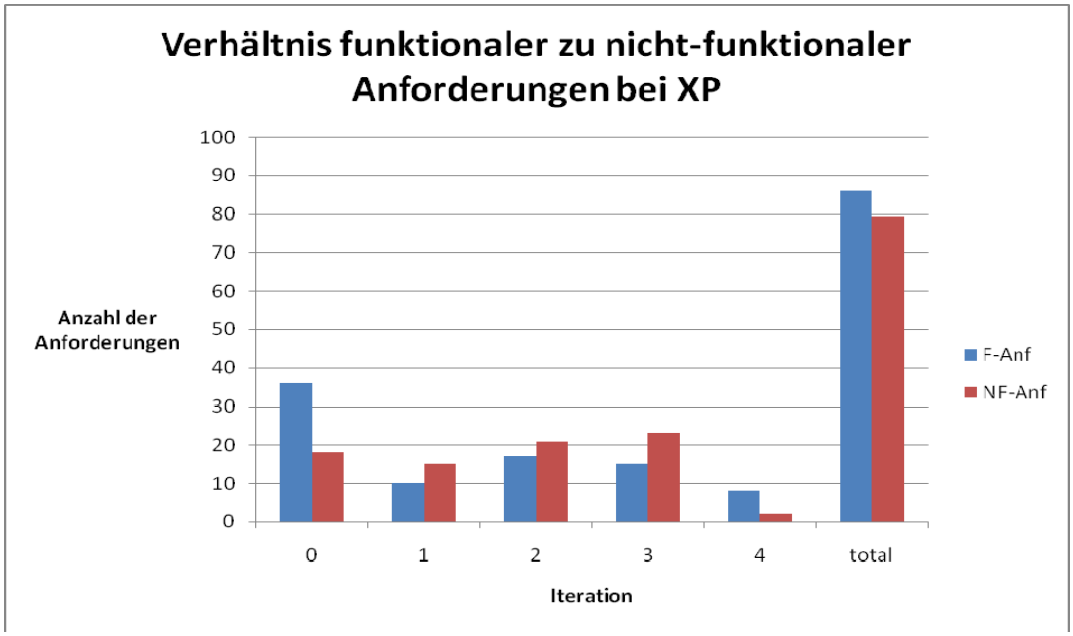

Abbildung 10: Verhältnis funktionaler zu nicht-funktionaler Anforderungen bei XP in P2

- G1.1.2:
  Untersuche das Verhältnis funktionaler zu nicht-funktionaler Anforderungen in Softwareprojekten

  Von den sechzehn ausgewerteten Softwareprojekten haben fünfzehn einen Anteil nicht-funktionaler Anforderungen kleiner oder gleich 45%. Ein Softwareprojekt hat einen Anteil von 65%. Im Durchschnitt liegt der Anteil der nicht-funktionalen Anforderungen aber nur bei 31%.

- G1.2.1:
  Untersuche das Verhältnis von Usability- zu anderen nicht-funktionalen Anforderungen in XP

  Mit einer Ausnahme (P2, Iteration 0) hatten die Usability-Anforderungen in beiden Projekten in allen Iterationen den größten Anteil an den nicht-funktionalen Anforderungen. Über den ganzen Projektzeitraum gesehen hatten die Usability-Anforderungen in P1 einen Anteil von 73% gemessen an allen nicht-funktionalen Anforderungen, in P2 einen Anteil von 68%.

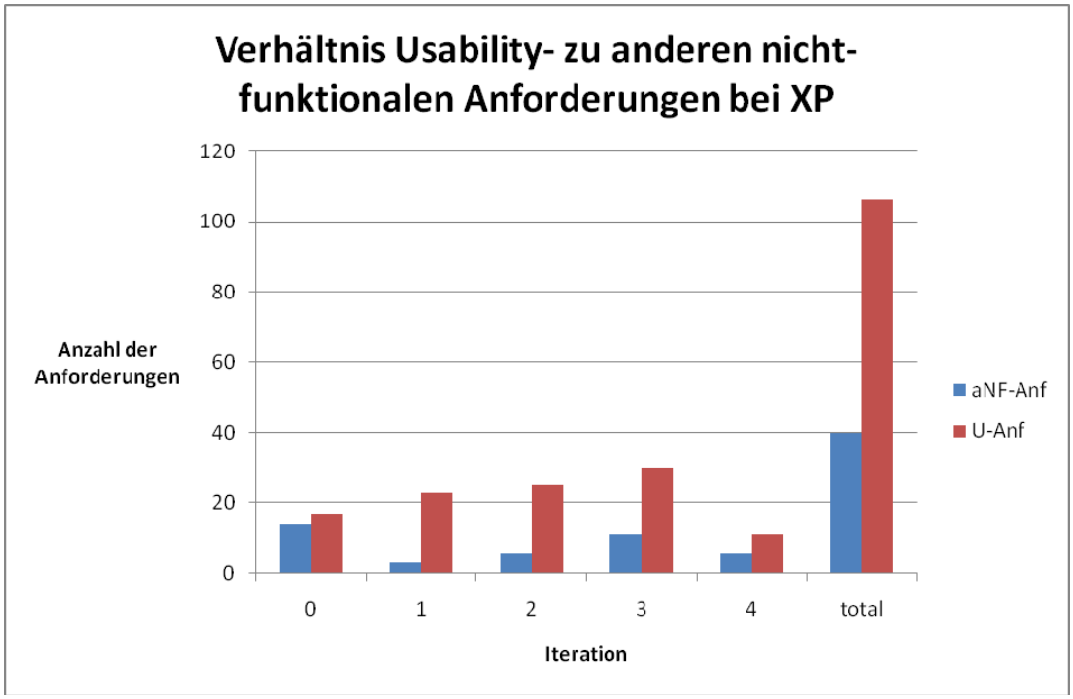

Abbildung 11: Verhältnis Usability- zu anderen nicht-funktionalen Anforderungen bei XP in P1

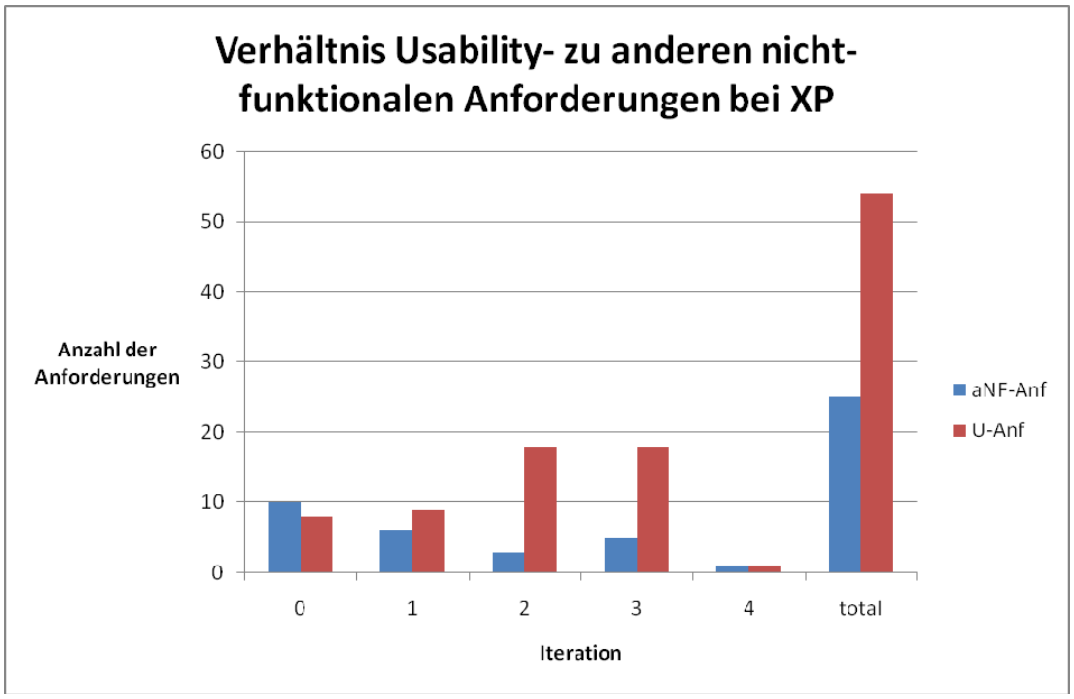

Abbildung 12: Verhältnis Usability- zu anderen nicht-funktionalen Anforderungen bei XP in P2

- G1.2.2:
  Untersuche das Verhältnis von Usability- zu anderen nicht-funktionalen Anforderungen in Softwareprojekten

  In den sechzehn ausgewerteten Softwareprojekten beträgt der Anteil der Usability-Anforderungen gemessen an allen nicht-funktionalen Anforderungen weniger als die Hälfte. Die Werte variieren stark zwischen 0 und 48%. Der durchschnittliche Anteil aller sechzehn Projekte beträgt 24%.

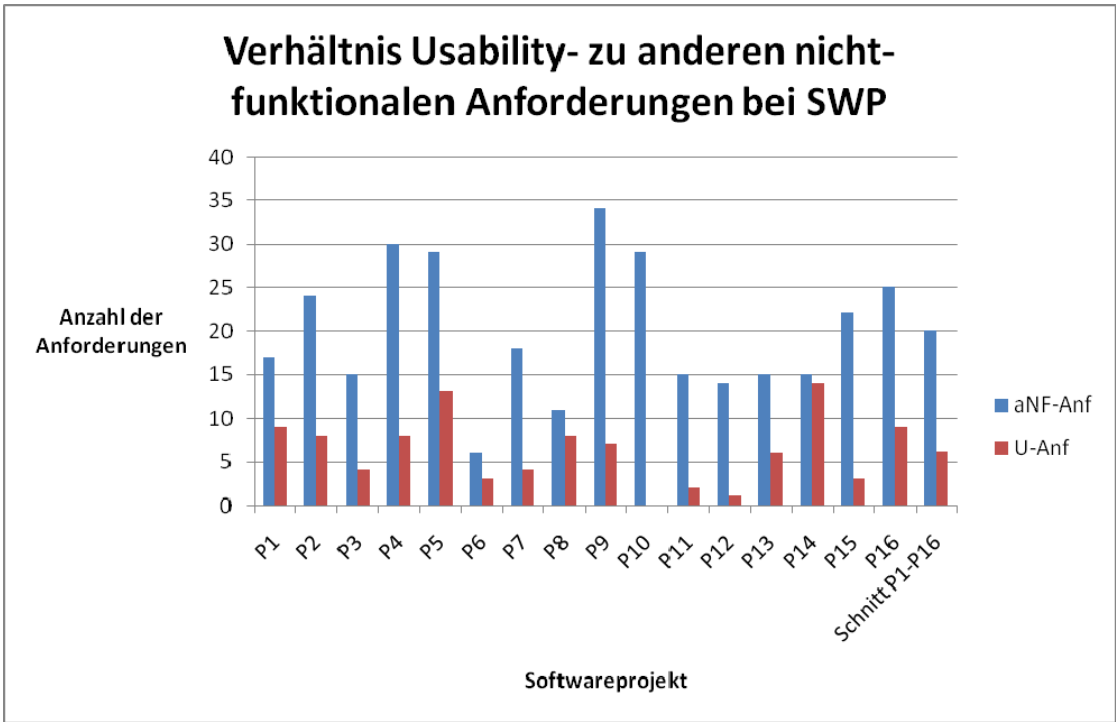

Abbildung 13: Verhältnis Usability- zu anderen nicht-funktionalen Anforderungen bei SWP

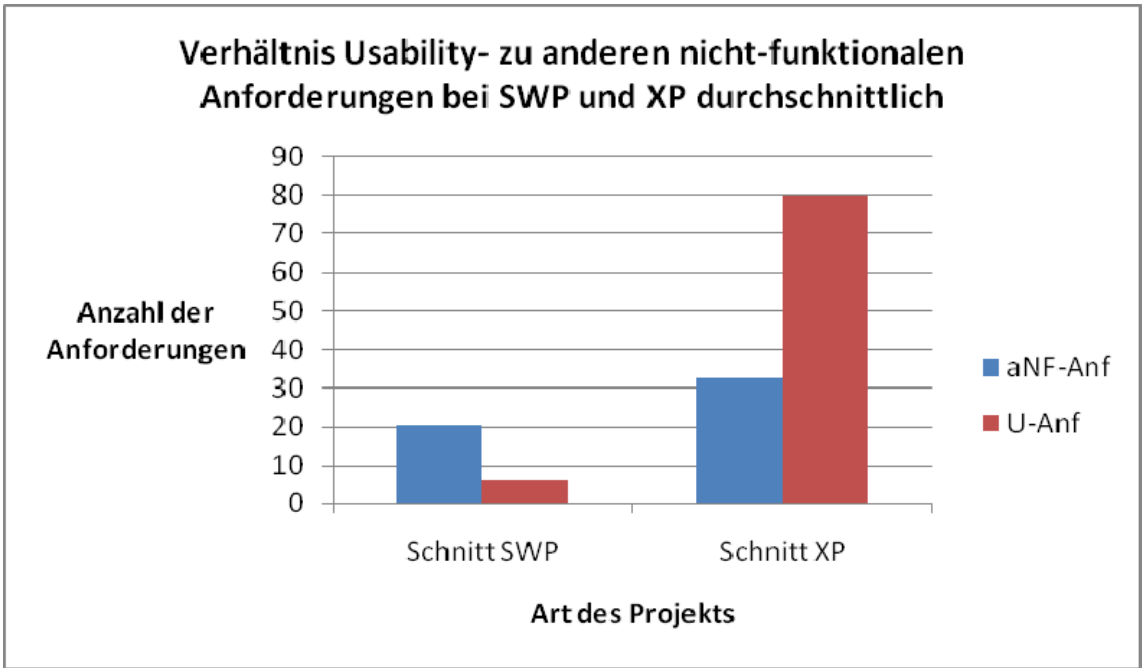

Abbildung 14: Verhältnis Usability- zu anderen nicht-funktionalen Anforderungen bei SWP und XP durchschnittlich

- G1.3:
  Ermittel das Verhältnis von Basisanforderungen zu später erhobenen Anforderungen

  In beiden Projekten wurde der geringere Teil der Anforderungen in Iteration 0 erhoben, mehr als die Hälfte der Anforderungen wurden erst in Iteration 1-4 hinzugefügt. In P1 machte der Anteil der in Iteration 1-4 hinzugefügten Anforderungen 65%, in P2 67% aus.

- G2.1:
  Messe den Grad der Änderungsanfälligkeit von Anforderungen in XP

  Nur jeweils 3 von 20 Story Cards in beiden Projekten blieben mit ihren Basisanforderungen in ihrer ursprünglichen Version erhalten. Alle anderen Story Cards waren Änderungen unterlegen bzw. wurden mit allen enthaltenen Anforderungen abgebrochen. Auffällig ist, dass gerade zu den Story Cards 1-4 im Laufe beider Projekte noch viele Anforderungen hinzugefügt wurden.

- G2.2:
  Messe den Grad der Granularität der Anforderungen in XP

  Die Messungen zeigen, dass sowohl innerhalb der Iterationen als auch über die ganze Projektdauer hin gesehen die feinkörnigen Anforderungen deutlich überwiegen. Die meisten Anforderungen handeln von messbaren Zielen und Subfunktionen. In Projekt 1 beträgt der Anteil dieser Anforderungen 80%, in Projekt 2 67%. Nur selten wird über Konkretes und Benutzerziele gesprochen (18-28%). Noch seltener wird über Abstraktes und Geschäftsziele gesprochen (2-5%).

5.3 Auswertung der Ergebnisse in Bezug zu den Hypothesen

Hypothese 1.1.:

Die Anzahl der nicht-funktionalen Anforderungen ist im Verhältnis zu den funktionalen Anforderungen bei XP höher als bei konventioneller Softwareentwicklung.

Die Hypothese wurde durch die Messung bestätigt. Während mit konventionellen Entwicklungsmethoden in den Softwareprojekten nur durchschnittlich 31% der Anforderungen nicht-funktional waren, kam man in den XP-Projekten zu einem Anteil von durchschnittlich mehr als 50%, wobei dieser Anteil in P1 bei 56% und bei P2 bei 48%liegt.

Natürlich variieren die Daten von Projekt zu Projekt. Es gibt in Projekten mit konventioneller Entwicklungsmethode Beispiele mit einem Anteil nicht-funktionaler Anforderungen von 45-65%. Da das aber bei den vorliegenden Daten nur bei zwei von sechszehn Projekten (12,5%) der Fall war, lässt sich vermuten, dass das die Ausnahme ist.

Es scheint durch die bessere Kommunikation bei XP, hauptsächlich bedingt durch den Kunden vor Ort, also tatsächlich viel mehr über die Eigenschaften der Software gesprochen zu werden. So gestaltet der Kunde das Programm schon während der Entwicklung mit anstatt unerwünschte oder fehlende Eigenschaften im Nachhinein bemängeln zu müssen. Dies ist ein Vorteil, der von XP erwartet wird, durch die Messung aber auch belegt werden konnte.

Hypothese 1.2.:

Die Anzahl der Usability-Anforderungen ist im Verhältnis zu anderen nicht-funktionalen Anforderungen bei XP höher als bei konventioneller Softwareentwicklung.

Während bei den XP-Projekten der Anteil der Usability-Anforderungen bei rund 70% liegt, kommt der Anteil in den konventionellen Softwareprojekten durchschnittlich nur auf 24%. Zu erwähnen ist, dass die Zahlen bei den Softwareprojekten zwar stark schwanken (0-48%), aber dennoch eine Tendenz zu erkennen ist, die dem Durchschnittswert sehr nahe kommt.
Der Kunde in XP hat eine viel bessere Chance, sich genaue Vorstellungen über sein gewünschtes Endprodukt zu machen. Frühzeitig gelieferte und testbare Software geben dem Kunden mehr Aufschluss darüber, welche Eigenschaften noch fehlen und welche vielleicht noch verbesserungswürdig oder überflüssig sind. Häufige Kommunikationsmöglichkeiten mit den Entwicklern ermöglichen es dem Kunden, diese Anforderungsänderungen auch zu vermitteln. In vielen kleinen Iterationen entwickelt sich das Produkt so stetig den Wünschen des Kunden entsprechend fort.

Bei konventionellen Softwareprojekten scheint es dem Kunden schwer zu fallen, den Entwicklern Hilfestellung bei der Berücksichtigung der Benutzerfreundlichkeit zu geben ohne vorher einen Prototypen begutachtet zu haben. Die Anforderungsspezifikation beinhaltet viel mehr technische Anforderungen auf der Seite der nicht-funktionalen Anforderungen.

Die Entwickler sind bei diesen klassischen Herangehensweisen den Aspekt der Benutzerfreundlichkeit betreffend viel mehr auf sich selbst angewiesen. Die Gefahr scheint groß, den Kunden am Ende nicht zufrieden stellen zu können, weil die Entwickler intuitiv

darüber entscheiden müssen, welche Eigenschaften dem Benutzer die Arbeit mit der Software erleichtert.

Hypothese 2.1.:

Die Usability-Anforderungen machen bei XP den größten Teil der nicht-funktionalen Anforderungen aus.

Die Hypothese konnte durch die Messung bestätigt werden. In den beobachteten Projekten waren mehr als die Hälfte der nicht-funktionalen Anforderungen Usability-Anforderungen.

Tendenziell scheint der Anteil von Usability-Anforderungen mit dem Fortschritt des Projektes größer zu werden. Diese Beobachtung konnte bei beiden gemessenen Projekten zumindest bis einschließlich zur dritten Iteration gemacht werden. Nur in der vierten Iteration konnte diese Annahme nicht bestätigt werden. Die Erklärung hierfür könnte sein, dass der Kunde in der verbleibenden Zeit möglichst viele der schon gestellten Anforderungen abgearbeitet sehen möchte, anstatt die Entwickler mit weiteren Anforderungen zu belasten, die dann wahrscheinlich nicht mehr bearbeitet werden können.

Wie bei der Herleitung der Hypothesen beschrieben, könnte der Grund dafür sein, dass dem Kunden schon früh brauchbare Software geliefert wird. Der Kunde kann die Software ausprobieren und sofort erkennen, welche Eigenschaften ihn stören oder ihm fehlen. So können die Bedürfnisse des späteren Benutzers der Software von Anfang an berücksichtigt werden. Dies setzt natürlich wieder die ausreichende Kompetenz des Kunden vor Ort voraus, diese Bedürfnisse den Entwicklern zu vermitteln.

Hypothese 2.2.:

Die meisten Anforderungen entstehen bei XP erst im Laufe des Projektes.

Auch diese Hypothese konnte bestätigt werden. Mehr als die Hälfte der Anforderungen wurde erst nach dem ersten Planning Game erhoben.

Es lässt sich allerdings keine Tendenz feststellen, die besagen könnte, dass zum Ende des Projektes die Anzahl der erhobenen Anforderungen grundsätzlich steigen würde. Es lässt sich aber sagen, dass die Möglichkeit, später Anforderungen hinzuzufügen, in jedem Fall in Anspruch genommen wird.

Die folgenden Kurven zeigen jeweils für ein Projekt die Anzahl der hinzugefügten Anforderungen pro Iteration.

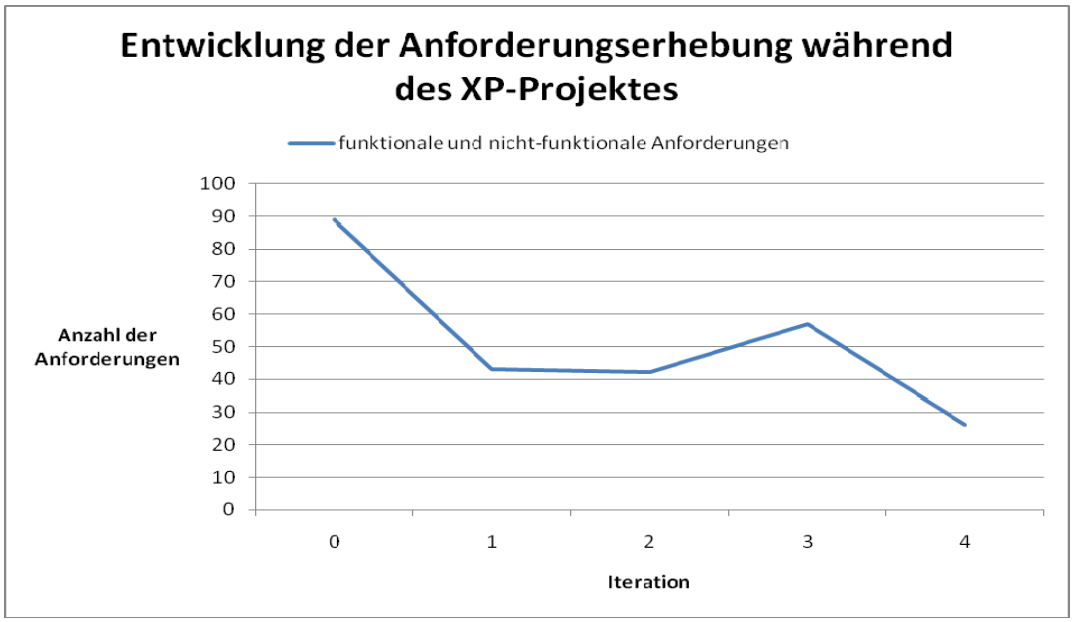

Abbildung 15: Anzahl der Anforderungen in P1

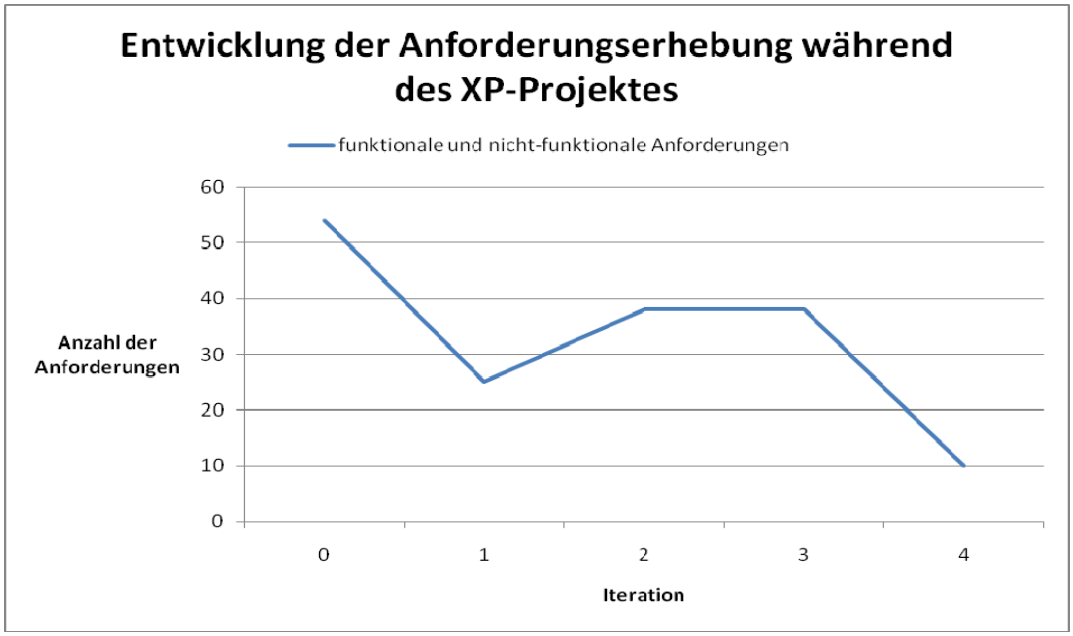

Abbildung 16: Anzahl der Anforderungen in P2

Eine Erklärung kann sein, dass eher skizzenhaften Aufzeichnungen der Basisanforderungen viele versteckte Anforderungen enthalten, die nicht explizit ausgesprochen und notiert wurden, aber die der Kunde in dem Zusammenhang mit der Story Card voraussetzt. Es ist auch denkbar, dass die eher skizzenhaften Aufzeichnungen gerade erst dadurch zustande kommen, dass der Kunde zu Beginn des Projektes selber noch gar nicht so genau weiß, wie die Details des Produkts aussehen sollen, und sie deshalb erst später im Projektverlauf den Entwicklern beschreiben kann.

Hypothese 2.3.:

Die Basis-Anforderungen bleiben meist nicht in ihrer ursprünglichen Version erhalten. Sie werden entweder während des Projekts geändert oder verworfen.

Jede Story Card enthält bei ihrer Erstellung eine gewisse Anzahl an Anforderungen, nämlich genau die, die bei der Erstellung der Story Card vom Kunden ausgesprochen wurden. Die Messung hat gezeigt, dass es in den meisten Fällen aber nicht bei dieser Anzahl an Anforderungen bleibt.

Beobachtungen während der Datenerhebung haben gezeigt, dass bei der Bearbeitung der Story Cards oder in Planning Games immer wieder Anforderungen hinzukommen. Diese können in vorher abstrakt formulierten Anforderungen enthalten gewesen sein. Möglich ist auch, dass sie vom Kunden vorausgesetzt wurden und erst bei der Story-Card-Abnahme zur Sprache kamen, als die dem Kunden vorgelegte Software nicht seinen Vorstellungen entsprach.

Die skizzenhaften Aufzeichnungen der Anforderungen bei XP kann also von Vorteil sein, solange dem Kunden in regelmäßigen Abständen brauchbare Software vorgelegt wird und Änderungswünsche von den Entwicklern angenommen werden.

Hypothese 2.4.:

In XP wird mehr über Subfunktionen und messbare Anforderungen gesprochen als über Geschäftsziele und abstrakte Anforderungen.

Die Messung zeigt, dass nur sehr wenig über abstrakte Anforderungen und Geschäftsziele gesprochen wird. Die meisten Anforderungen sind messbar oder handeln von Subfunktionen. Die Hypothese hat sich also bestätigt. Die Ergebnisse sind in den folgenden zwei Abbildungen grafisch dargestellt.

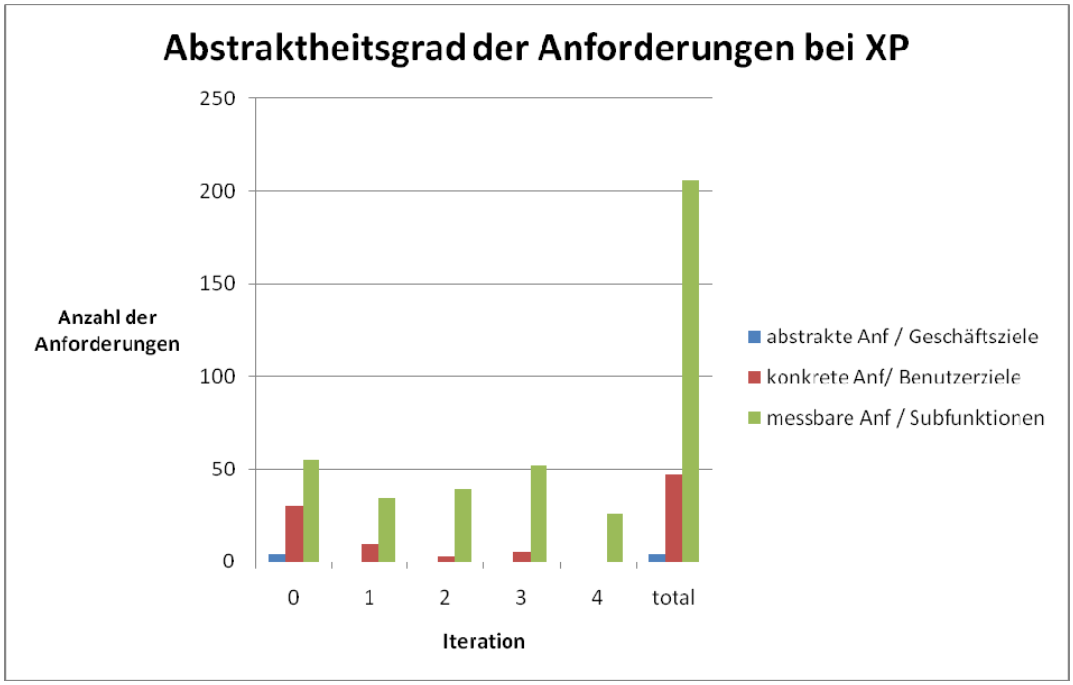

Abbildung 17: Abstraktheitsgrad der Anforderungen bei XP in P1

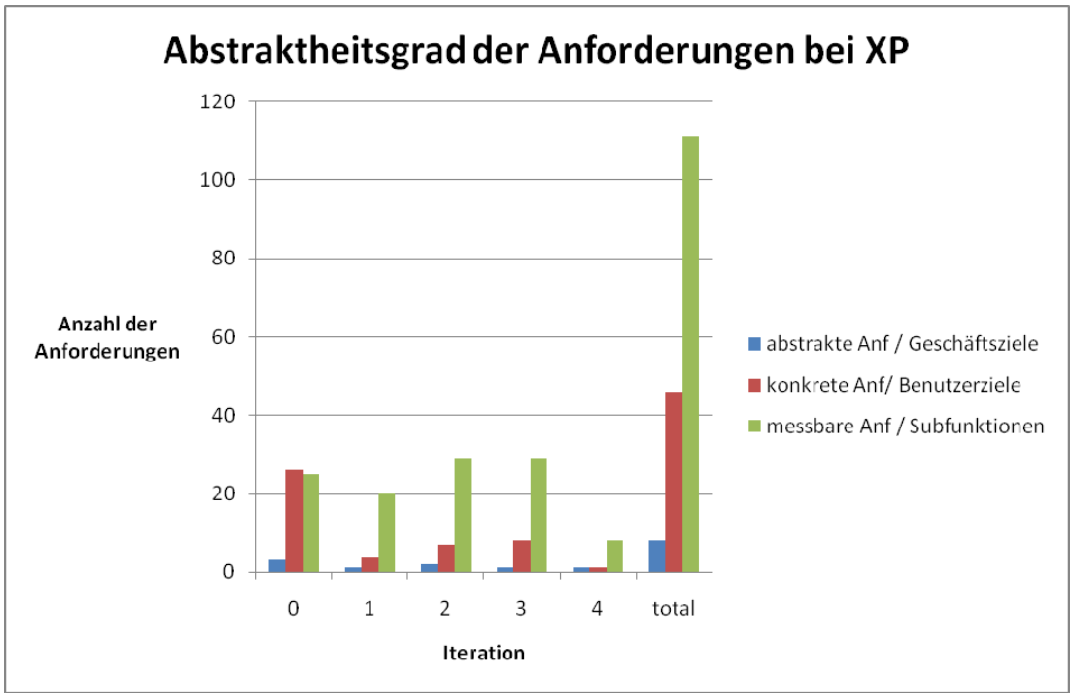

Abbildung 18: Abstraktheitsgrad der Anforderungen bei XP in P2

Die häufige Kommunikation zwischen dem Kunden und den Entwicklern und die frühe Erstellung brauchbarer Software scheinen auch hier wieder ausschlaggebend zu sein. Dadurch, dass der Kunde schon früh Software zum „Anfassen“ bekommt, kann er sich ein genaueres Bild von dem gewünschten Endprodukt machen. Diese Details waren dem Kunden anfangs vielleicht noch gar nicht klar vor Augen. Häufige Gespräche lassen die Anforderungen außerdem feinkörniger werden.

6 Bewertung der Gültigkeit

6.1. Bewertung der Metriken

Wie erwartet war die Klassifizierung der Anforderungen nicht immer eindeutig. Bei beiden Protokollantinnen zeigten sich Unsicherheiten. In P1 stufte die Protokollantin rund 10% der klassifizierten Anforderungen als unsicher ein, in P2 waren es 4%. Wie schon in Kapitel 3.2 gesagt, werden diese Unsicherheiten im Rahmen dieser Studienarbeit toleriert, da sie im Rahmen der erwarteten 10% geblieben sind.

Sieht man von den Unsicherheiten in der Klassifizierung ab, lässt sich grundsätzlich zu den Metriken sagen, dass ihre Anwendung zu einfachen und gut analysierbaren Ergebnissen geführt hat. Die Ergebnisse konnten gut in Prozentzahlen wiedergegeben und dann miteinander verglichen werden.

6.2 Bewertung des Beobachtungsgegenstandes

Alle Projekte wurden im studentischen Umfeld durchgeführt. Hier sind folgende Unterschiede zu Projekten in der Realität zu berücksichtigen:

- Möglich ist eine geringere Motivation der Teilnehmer, da Fehler in der Entwicklung nicht den Arbeitsplatz gefährden. Die Atmosphäre ist weniger ernsthaft und kann zu einem mangelndem Engagement der Studenten führen. Diese Auswirkungen des universitären Umfelds auf die Studenten haben sich in den beobachteten XP-Projekten nicht gezeigt.

  Optimierung: Reale Kunden könnten die Motivation erhöhen.

- In der beobachteten Lehrveranstaltung hatte der Kunde häufig zugleich auch eine beratende Funktion. Bei der Beantwortung von Fragen in der Rolle als Veranstaltungsleiter steht er oft in einem Konflikt zu seiner Rolle als Kunde. Er muss genau abwägen, wie viel Hilfestellung er den Studenten geben kann, ohne die Projektergebnisse zu beeinflussen.

  Optimierung: Der Veranstaltungsleiter in der Rolle des Kunden könnte von seinen organisatorischen Pflichten im Rahmen der Veranstaltung befreit werden.

- Es stellt sich die Frage, ob die Anzahl von zwei XP-Projekten und sechszehn Softwareprojekten wirklich ausreichend ist, um repräsentative Ergebnisse und endgültige Antworten auf die Hypothesen zu liefern.

  Optimierung: Einerseits könnten weitere Projekte beobachtet und analysiert werden, andererseits wäre es auch interessant zu sehen, ob die gleiche Aufgabe von einem anderen Team ähnlich gelöst wird und die Ergebnisse bezogen auf die Messung in dieser Studienarbeit ähnlich sind.

6.3 Bewertung der Abweichungen der Ergebnisse

Die Ergebnisse aus den Messungen der beiden XP-Projekte zeigen einen deutlichen Unterschied zwischen den Projekten.

- In P1 gab es etwa hundert Anforderungen mehr als in P2.
- In P1 überwiegen die nicht-funktionalen Anforderungen, in P2 (trotz einem hohen Anteil nicht-funktonaler Anforderungen) die funktionalen.

Wie kann es zu diesen Unterschieden kommen?

In P1 wurde das Ressourcenplanungs-Tool entwickelt. In diesem Projekt wurden viel mehr technische Details besprochen und solche, die die GUI betreffen. Der Kunde gab genaue Anforderungen zum Design der Software. In P2, das sich mit dem Webservice beschäftigte, war das Design dagegen zunächst von schwächerer Priorität. Hier gab es in der 0. Iteration Schwierigkeiten mit der Umsetzung der ersten Story Cards. In der 1. Iteration wurde das Projekt neu gestartet. Die Entwickler konnten nach dem Neustart noch nicht so viel Zeit in das Design stecken wie die andere Entwicklergruppe in ihrem Projekt. Der Anteil nicht-funktionaler Anforderungen, der die GUI und die technische Umsetzung betrifft, scheint genau die oben genannten Unterschiede auszumachen. Diese Hypothese ließe sich mithilfe der Messdaten untersuchen, indem die technischen und die GUI betreffenden Anforderungen in P1 ausgezählt würden und mit der Differenz zwischen den Anforderungen in P1 und denen in P2 verglichen würden. Im Rahmen dieser Studienarbeit wurde aber aus Zeitgründen auf die Überprüfung dieser Hypothese verzichtet.

6.4 Bewertung des Einflusses unterschiedlicher Erfasser

Zur Abschätzung der Wiederholbarkeit der Datenerhebung wurde ein Projekt teilweise doppelt erfasst. Die Daten wurden von den jeweiligen Erfassern klassifiziert. Die Klassifikationen wurden miteinander verglichen. Es stellte sich heraus, dass sich die Klassifizierungen nur geringfügig voneinander unterschieden, der Einfluss auf die Ergebnisse war unwesentlich.

7 Fazit und Ausblick

Aufgabe dieser Studienarbeit war es, Anforderungen bei agilen und konventionellen Softwareprojekten untersuchen und vergleichen. Dafür wurde eine Messung in einem universitären Umfeld mithilfe der GQM-Methode geplant und durchgeführt.

Die Hypothesen zu dem Vergleich der unterschiedlichen Vorgehensmodelle konnten bestätigt werden. Die Ergebnisse entsprachen überwiegend den Erwartungen. So sind nicht-funktionale Anforderungen, insbesondere Usability-Anforderungen, in agilen Softwareprojekten von größerer Bedeutung als in konventionellen. Der Vergleich zeigte außerdem, dass der Kunde in agilen Vorgehensmodellen mit dem Fortschritt des Projekts immer mehr über Details spricht, nachdem die Anforderungen an das Produkt anfangs nur skizziert wurden. Es scheint, als würden die nach der nullten Iteration hinzugefügten Anforderungen größtenteils Verfeinerungen der ersten geschriebenen Story Cards sein, die der Kunde durch die Begutachtung von Prototypen immer genauer beschreiben kann. Es konnte im Rahmen dieser Studienarbeit allerdings nicht abschließend geklärt werden, ob die nach der nullten Iteration hinzugefügten Anforderungen hauptsächlich detaillierte Informationen zu früh erstellten Story Cards sind oder ob es andere Gründe dafür gibt, dass die meisten Anforderungen erst im Laufe des Projekts entstehen. Um diese Aussage bestätigen zu können, dürfte man sich nicht ausschließlich auf die Anzahl der hinzugefügten Anforderungen und die Anzahl der Anforderungen über Subfunktionen und Messbares berufen, sondern müsste jede Anforderung über Subfunktionen und Messbares inhaltlich einer Story Card zuordnen können. Da die Zuordnung aber nicht immer klar ist, müsste in diesem Fall auch der Kunde mit einbezogen und zu den Anforderungen befragt werden. Dabei müsste sicher gestellt werden, dass der Kunde durch die Fragen in seinen Entscheidungen bzgl. der Anforderungen nicht beeinflusst wird. Der Messplan müsste für ein solches weiterführendes Forschungsprojekt modifiziert werden. Die Durchführung könnte erst in einem nächsten XP-Labor erfolgen.

In weiteren Forschungsprojekten um das Thema Softwareentwicklung wäre es vielleicht interessant zu untersuchen, ob die vielen Gespräche über Details in agilen Softwareprojekten vielleicht auch daher rühren, dass während der Erstellung der Story Cards Anforderungen zunächst unausgesprochen bleiben, weil der Kunde sie entweder selber noch nicht kennt oder aber sie im Zusammenhang mit den in der Story Card skizzierten Anforderungen voraussetzt. Die erfassten Anforderungen können mit großer Wahrscheinlichkeit nicht alle Kundenwünsche abdecken. Dennoch ist es möglich, dem Kunden ein gutes, seinen Wünschen entsprechendes Produkt auszuliefern - durch gute Kommunikation während der Produktentwicklung.

Beide Arten der Softwareentwicklung zeigen Vor- und Nachteile und beide bieten Lösungsansätze, wie die Auswirkungen der Nachteile gering gehalten werden können. Es muss je nach Projektart (Vertragsbedingungen, Verfügbarkeit des Kunden, Zeitvorgabe usw.) entschieden werden, welche Entwicklungsmethode am sinnvollsten ist oder ob man die für das Projekt wichtigsten Eigenschaften der verschiedenen Entwicklungsmethoden vielleicht sogar kombinieren kann, um die Kundenwünsche bestmöglich zu erfüllen. Eine Möglichkeit bei strengen Vertragsbedingungen, die auf einer Anforderungsspezifikation beruhen sollen, wäre es zum Beispiel sich für ein konventionelles Vorgehensmodell zu entscheiden, aber mehr Energie in die Kommunikation zwischen dem Kunden und den Entwicklern zu stecken, damit Änderungswünsche früh erkannt werden und ohne einen zu großen finanziellen Aufwand noch berücksichtigt werden können. Andersherum wäre es doch denkbar, die in konventionellen Vorgehensmodellen sehr ausführliche Dokumentation in reduzierter Form zu

einem Teil einer agilen Entwicklungsmethode zu machen, um dem Kunden etwas mehr Verbindlichkeit der Anforderungen als Sicherheit für den Erfolg des Projekts zu geben.

8 Quellenverzeichnis

[Bec+AM] http://www.agilemanifesto.org
Kent Beck +
Manifesto for Agile Software Development
eingesehen am 06.07.2008

[Bec+AP] http://www.agilemanifesto.org/principles.html
Kent Beck +
Principles behind the Agile Manifesto
eingesehen am 06.07.2008

[Bec+00] Kent Beck and Martin Fowler
Planning Extreme Programming
Addison-Wesley, 2000

[Bec+04] Kent Beck and Cynthia Andres
Extreme Programming Explained
Addison-Wesley, 2004

[Bec00] Kent Beck
Extreme Programming
Addison-Wesley, 2000

[Boeh88] Barry Boehm
A spiral model of software development and enhancement
IEEE Computer, vol.21, no. 5, Mai 1988, S.61-72

[Boeh02] Barry Boehm
Get ready for Agile Methods, with Care
2002

[Bou08] Christian El Boustani
Bachelorarbeit
Quantitative und qualitative Messung von Software-Anforderungen
28.01.2008

[Jef01] http://www.xprogramming.com/xpmag/whatisxp.htm
Ron Jeffries
What is Extreme Programming?
eingesehen am 06.07.2008, 2001

[LL06] Ludewig und Lichter
Software Engineering–Grundlagen, Menschen, Prozesse, Techniken
dpunkt Verlag, 2006

[LS05] Daniel Lübke, Kurt Schneider
Agil Hour Teaching XP Skills to Students and IT-Professionals
Springer Verlag, 2005

[Rup06] Chris Rupp
Requirements Engineering und -Management
Hanser, 2006

[SB99] Rini van Solingen, Egon Berghout
The Goal/Question/Metric Method.
McGraw-Hill Publishing Company, 1999

[Schn07] Kurt Schneider
Abenteuer Software Qualität
dpunkt Verlag, 2007

[Schn05] Kurt Schneider
Vorlesung: Softwaretechnik WS 05/06
2005

[Schw04] Ken Schwaber
Agile Project Management with Scrum.
Microsoft Press, 2004

[ZGK04] Wolfgang Zuser, Thomas Grechenig, Monika Köhle
Software Engineering mit UML und dem Unified Process
Pearson Studium, 2004