Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Prozessanalyse Nr. 3

Analyse des Entwicklungsprozesses bei Softwareentwicklern

Verena Käfer, Florian Gänßlen, Burak Incearap

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Wagner
Betreuer:	Dipl.-Ing. Jan-Peter Ostberg
begonnen am:	04.12.2013
beendet am:	04.06.2014
CR-Klassifikation:	K.6.3

Kurzfassung

Im Rahmen unserer Prozessanalyse an der Universität Stuttgart haben wir den Arbeitsablauf eines Softwarehauses in Ludwigsburg analysiert und verbessert. Der Entwicklungsprozess basiert auf dem Vorgehensmodell Scrum. Um die eigentliche Situation besser zu verstehen, haben wir uns die vorliegenden Abläufe erklären lassen. Dabei sind uns an verschiedenen Stellen Probleme und Unregelmäßigkeiten aufgefallen.
Im Rahmen der Analyse haben wir deshalb für den Unterprozess des Testens Verbesserungsvorschläge erstellt. Neben der Erklärung von verschiedenen Testabläufen, die in die bestehende Struktur integriert werden können, stellen wir auch ein Werkzeug für automatische Systemtests vor.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Aufgabenstellung

Unter einer Prozessanalyse versteht man die Untersuchung und Überprüfung von einem oder mehreren Prozessen bzw. Abläufen sowie die Zerlegung dieser in ihre Einzelteile. Anhand dieser Analyse des „Ist-Zustands“ können dann konkrete Verbesserungen für einen „Soll-Zustand“ vorgeschlagen werden.

Im Master-Studiengang Softwaretechnik der Universität Stuttgart wird solch eine Prozessanalyse in einer Gruppe bestehend aus drei Personen durchgeführt. Als Partner für diese Aufgabe hatte sich ein kleines und noch relativ junges Softwareunternehmen aus Ludwigsburg zur Verfügung gestellt. Es wurde der allgemeine Softwareentwicklungsprozess analysiert und dann festgelegt, dass der Teilprozess des Testens näher untersucht und verbessert werden sollte.

1.2 Vorgehen

Um den Gesamtprozess zu verstehen, haben wir uns mehrere Male mit den Mitarbeitern unseres Prozesspartners getroffen. Dabei haben wir sowohl mit dem Geschäftsführer als auch mit den Teammitgliedern über die derzeitige Situation gesprochen. Uns wurde der Gesamtprozess erklärt und wir konnten bei mehreren Meetings und Kundenbesuchen anwesend sein. Um ein besseres Verständnis für die vorliegende Situation zu bekommen, haben wir verschiedene Fakten zusammengetragen und diese mit den Verantwortlichen noch einmal besprochen. Dabei sind uns verschiedene Probleme aus unterschiedlichen Bereichen und Abläufen aufgefallen, die wir daraufhin visualisiert und in einer Präsentation vorgestellt haben. Die meisten Probleme konnten wir im Bereich des Testens feststellen, wofür wir mögliche Verbesserungen erarbeitet haben und in dieser Ausarbeitung vorstellen werden.

2 Ist-Analyse

Um vorhandene Probleme im Prozess unseres Partners zu finden, ist es wichtig gewesen, zunächst den Gesamtprozess zu verstehen. Dafür wurde insbesondere die spezielle Entwicklungssituation miteinbezogen.

2.1 Beschreibung des Projektpartners

Bei unserem Projektpartner handelt es sich um ein kleines Team von jungen Softwareentwicklern. Alle Mitarbeiter sind an der Entwicklung von Software beteiligt. Dies führt dazu, dass alle Entwickler in einem Raum sitzen und eine sofortige Kommunikation somit jederzeit möglich ist.

2.2 Grundlagen des Prozesses

Der Gesamtprozess ist Scrum-basiert[1] und sehr agil. Es wird viel Wert auf die Zusammenarbeit mit dem Kunden gelegt. Aufgrund der kleinen Mitarbeiterzahl werden nicht alle Scrum-Rollen vollständig übernommen. Somit werden mehrere Rollen in einer Person vereint. Neben einem großen Projekt, das die Hauptarbeit darstellt, gibt es auch noch verschiedene kleinere, die parallel dazu durchgeführt werden.

2.3 Ablauf

Im folgenden Abschnitt wird der Gesamtprozess in einzelnen Schritten vorgestellt. Abbildung 2.1 stellt diese grafisch in ihrer Reihenfolge dar. Zu jedem Teilschritt erfolgt eine genaue Erklärung sowie die von uns festgestellten Probleme.

[1] http://de.wikipedia.org/wiki/Scrum

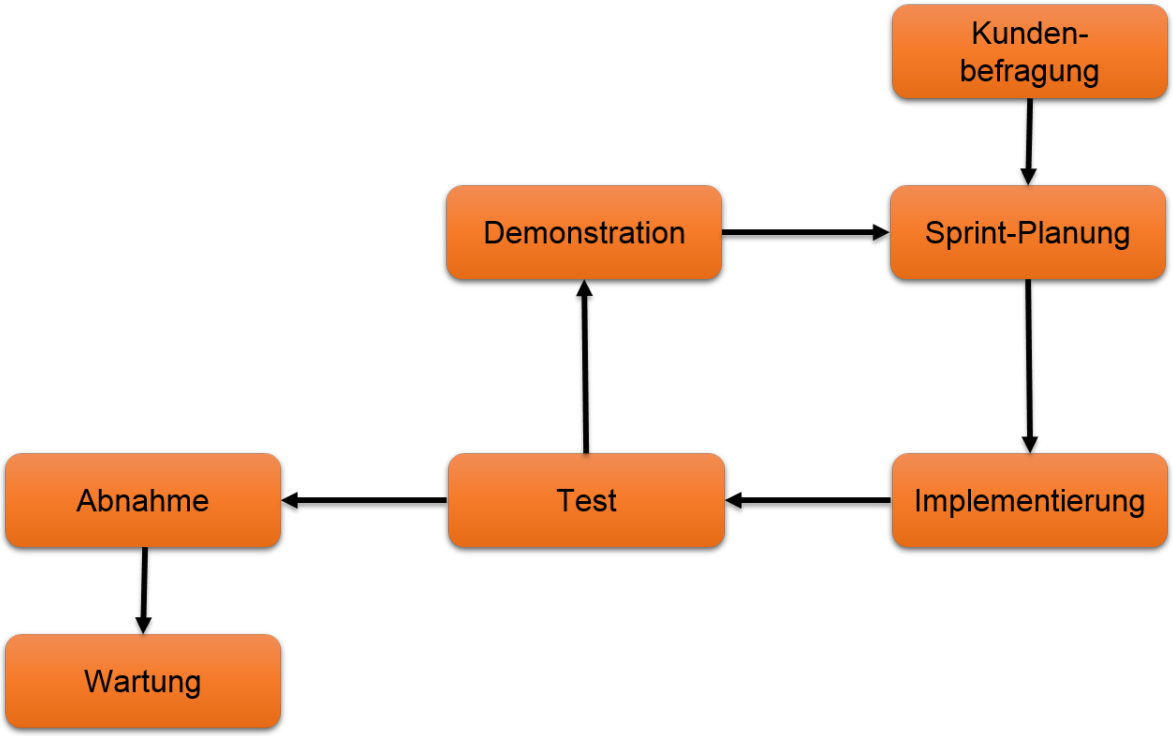

Abbildung 2.1: Allgemeiner Ablauf eines Projekts

2.3.1 Kundenbefragung und weitere Schritte

Die Kundenbefragung findet zu Beginn jedes Projekts statt. Der Kunde erklärt dabei, wie er sich die fertige Software vorstellt und es werden die Anforderungen besprochen und aufgenommen. Die genannten Anforderungen werden dabei jeweils in einer User-Story festgehalten. Daraufhin folgt die Planung des ersten Sprints. Dieser wird implementiert und getestet. Die erzielten Ergebnisse werden dem Kunden in einer Demonstration vorgestellt und von diesem abgenommen. Die Software wird dabei nach jedem Sprint ausgeliefert. Dadurch kann es während des weiteren Projektablaufs bereits zu Wartungsarbeiten kommen. Der vorhandene Kreislauf wird mit dem Beginn des nächsten Sprints geschlossen. Da bisher noch kein größeres Projekt beendet wurde, können noch keine Angaben bezüglich der letzten Abnahme und der späteren Wartung gemacht werden.

Festgestellte Probleme

Bei dieser ersten Befragung werden nur die offensichtlichen Anforderungen erhoben. Seltene Anforderungen werden nicht angesprochen oder erst später erkannt. Dadurch ist das Anforderungsdokument unvollständig.

Zudem gibt es keinen Projektplan. Geplant wird von Sprint zu Sprint. Auch eine ausformulierte Spezifikation ist nicht vorhanden. Alle Anforderungen werden nur als User-Story festgehalten.

2.3.2 Sprint-Planung

Jeder Sprint beginnt mit einer Sprint-Planung. In dieser wird mit dem Kunden diskutiert, welche User-Stories im Sprint bearbeitet werden sollen. Der Kunden priorisiert dabei die User-Stories. Die User-Stories werden daraufhin vom Team in Unteraufgaben aufgeteilt. Dabei wird auch über die grobe technische Umsetzung diskutiert. Welche Komponenten müssen geändert werden? Wo könnte es Schwierigkeiten geben? Jede Aufgabe wird mit einem Akzeptanzkriterium versehen.
Anschließend wir mit Hilfe eines Planungs-Poker der Aufwand jeder Aufgabe geschätzt.
Einmal im Monat geht ein Dokument mit allen bisher erhobenen Anforderungen zum Kunden, welcher dieses handschriftlich ergänzt und dann zurückschickt.

Festgestellte Probleme

Die Aufteilung der User-Stories in einzelne Aufgaben geschieht über ein separates Programm, in welches die Stories manuell kopiert und dann aufgeteilt werden. Dabei kommt es hin und wieder zu Übertragungsfehlern in Form von falschen oder doppelten Zuweisungen zu Arbeitspaketen. Aufgrund der fehlenden bzw. unvollständigen Dokumente kommt es in seltenen Fällen zu Unklarheiten bezüglich des Umfangs einer User-Story, dann muss zuerst eine Rücksprache mit dem Kunden für Klarheit sorgen.

2.3.3 Implementierung

Bei der Implementierung werden nacheinander die Aufgaben des jeweiligen Sprints abgearbeitet. Dabei stellen die Kommentare im Quellcode die Dokumentation dar. Kleinere Projekte, die nur von einer Person durchgeführt werden, finden ohne Diskussion und Kontrolle statt.

Festgestellte Probleme

Es gibt Entwurfs- und Spezifikationsdokumente, welch die Kernmodule oder wichtige Algorithmen zeigen. Ein Großteil der Dokumentation findet jedoch im Quell-Code statt. So ist es schwierig, manche Dinge direkt nachzulesen. Es gibt keine weiteren Dokumente, welche einen Ablauf und somit die nächsten Schritte vorgeben könnten. Auch ein allgemeiner Zielpunkt ist nicht wirklich definiert. Diese Missstände sorgen dafür, dass die Implementierungsphase innerhalb eines Sprints ohne richtigen Plan verfolgt wird. Sollte das Projekt einmal zur Seite gelegt und zu einem späteren Zeitpunkt wieder aufgenommen werden, so wird eine Weiterführung schwerfallen, da die erklärenden Dokumente fehlen und evtl.

Personen mit Wissen darüber nicht mehr vorhanden sind. In Abbildung 2.2 ist dieser Kreislauf einmal dargestellt. Erschwerend kommt hinzu, dass dieser Ablauf durch immer neue Kunden-Anforderungen ständig verschärft wird. Somit wird die bereits unvollständige Dokumentation noch weniger gepflegt, bis sie irgendwann gar nicht mehr beachtet wird. Sollten diese Umstände eines Tages trotzdem ein funktionierendes Produkt hervorgebracht haben, so wird auch die Wartung nach Beendigung des Projekts nur schwer und unzureichend möglich sein.

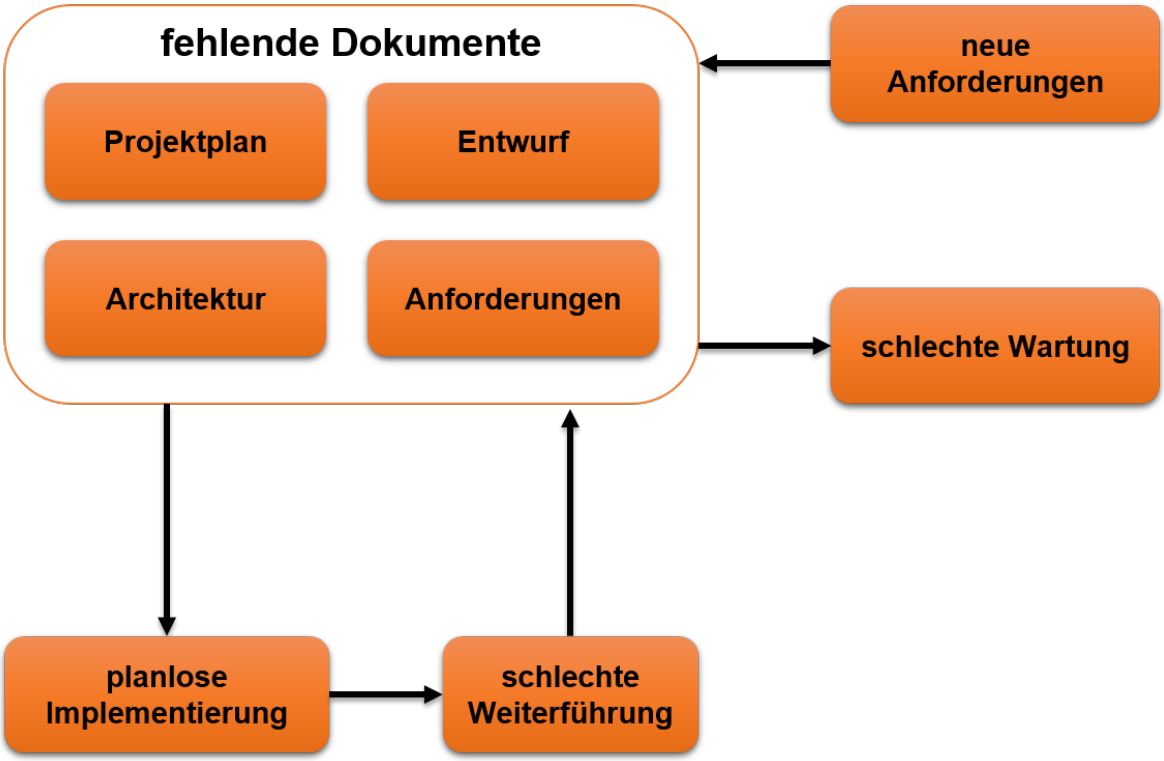

Abbildung 2.2: Festgestellter Problemkreislauf im Bereich der Implementierung

2.3.4 Test

Nach jedem Commit in ein Versionsverwaltungssystem werden Unit-Tests durchgeführt. Es findet eine kontinuierliche Integration in das bereits vorhandene System statt. Hier wurde auch der Wunsch automatischer Systemtests geäußert. Nach Abschluss einer User-Story wird diese nach ihrer Beschreibung getestet. Es gibt keinen Testplan, der auch eventuelle Sonderfälle überprüfen würde.

Festgestellte Probleme

Es kommt gerade bei kleinen Projekten vor, dass ein Entwickler seinen eigenen Code testet. Dies sollte eigentlich nur im äußersten Notfall durchgeführt werden, wird hier aber als Regel betrachtet. Des Weiteren gibt es keine spezifizierten Testfälle und es werden keine Reviews durchgeführt. Der Tester muss bei seiner Arbeit an etwaige Sonderfälle denken und diese überprüfen.

Wie bereits erwähnt, gibt es keinen vollständigen Systemtest. Jede User-Story wird einmal nach ihrer Fertigstellung getestet. Nach dem Hinzufügen neuer Klassen und Komponenten werden nur noch die User-Stories überprüft, die nach der Meinung und Erfahrung des Entwicklers nun einen neuen Fehler enthalten könnten. Die Hauptquelle für aufgetretene Fehler ist der Kunde. Er wird hier also unbeabsichtigt als Tester herangezogen und muss vorhandene Fehler melden, damit diese behoben werden können.

2.3.5 Kundendemonstration

Nach jedem Sprint erfolgt die Präsentation der neuen Funktionen beim Kunden. Anschließend werden die Inhalte des nächstens Sprints besprochen und diskutiert. Hierbei ist es möglich, dass neue Anforderungen aufgenommen werden, die dem Kunden bei der Demonstration eingefallen sind.

Festgestellte Probleme

Es werden immer wieder neue Anforderungen aufgenommen. Dadurch lässt sich der Gesamtaufwand für ein Projekt schlecht abschätzen und ein möglicher Ziel-und Endpunkt wird damit immer weiter nach hinten geschoben.

2.3.6 Kundenabnahme

Bei der Kundenabnahme handelt es sich um die finale Präsentation vor dem Kunden. Der Ablauf für kleine Projekte war bisher ähnlich zu einer Sprint-Demonstration. Es wurde alle Bestandteile der entwickelten Software vorgestellt und vorgeführt. Für ein großes Projekt gab es bisher keine endgültige Kundenabnahme.

2.3.7 Wartung

Bei bisherigen Projekten erfolgte die Wartung durch Besuche beim Kunden, um zu sehen, ob die Software wie gewünscht funktioniert. Da bisher kein größeres Projekt abgeschlossen wurde, können hier keine weiteren Angaben gemacht werden.

Festgestellte Probleme

Da es nur wenig Dokumentation außerhalb des Codes gibt, stellen wir uns die Wartung eines größeren Projekts unter diesen Umständen als schwierig vor. Die dafür zuständige Person muss den Code sehr gut kennen, um die passende Stelle für Veränderungen und Fehlerbehebungen zu finden. Bei neuen und mit der Materie nicht vertrauten Personen kann es deshalb zu Problemen und einer langen Einarbeitungszeit kommen. Sollte einmal ein Mitarbeiter das Unternehmen verlassen, so geht auch gleichzeitig Wissen über verschiedene Bereiche verloren.

2.3.8 Fazit

Im Großen und Ganzen funktioniert der allgemeine Entwicklungsprozess, an verschiedenen Stellen sind aber Anpassungen und Veränderungen nötig. Durch die räumliche Nähe und die kleine Teamgröße funktioniert vieles, was bei einem größeren Team nicht funktionieren würde. Die Hauptprobleme sehen wir bei der unvollständigen Dokumentation und dem vorhandenen Testablauf.
Nach Absprache mit unserem Projektpartner haben wir uns deshalb dazu entschlossen, den Unterprozess des Testens mit Vorschlägen und Anmerkungen zu verbessern. Die entstandene Lösung wird in Kapitel 3 vorgestellt.

3 Unser Verbesserungsvorschlag

In diesem Kapitel stellen wir unsere Verbesserungsvorschläge vor. Es wird zuerst ein verbesserter Scrum-Prozess nach [GG12] vorgestellt und beschrieben, wie er von unserem Prozesspartner umgesetzt werden kann. Zusätzlich wird ein Werkzeug für automatische Systemtests vorgeschlagen.

3.1 Erweiterter Scrum-Prozess

In Abbildung 3.1 ist der Ablauf des bekannten Scrum-Prozesses zu sehen. Dieser wird bis auf einige Ausnahmen auch so von unserem Projektpartner verwendet.

Am Anfang eines jeden Projekts werden die Anforderungen an das Produkt besprochen und in einem Backlog verwaltet. Für jeden Sprint findet eine eigene Sitzung statt, in welcher die zu erledigenden Aufgaben in ein kleineres Sprint-Backlog übertragen werden. Täglich findet ein Scrum-Meeting statt, um über den aktuellen Stand zu sprechen. Das Ergebnis eines Sprints ist ein Inkrement, dass direkt vom Kunden verwendet werden kann. Nach dem letzten Sprint wird die eigentliche Auslieferung für das fertige Produkt geplant.

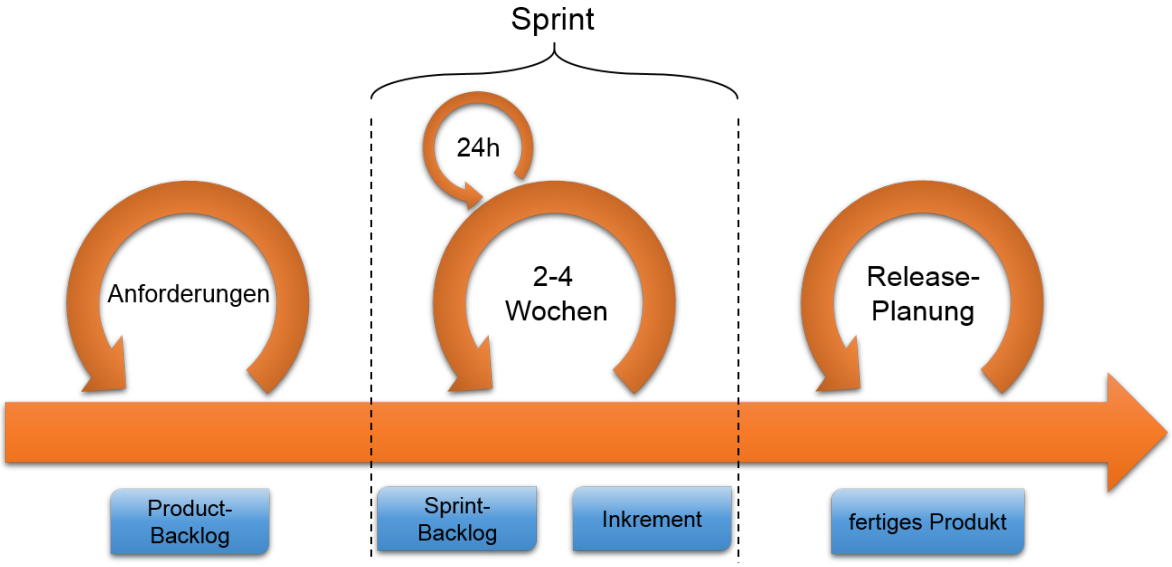

Abbildung 3.1: Die einzelnen Abläufe und Phasen in Scrum

Um den Testprozess zu verbessern, schlagen wir nach [GG12] einige Veränderungen vor. Diese sind in Abbildung 3.2 grafisch dargestellt und erweitern den bestehenden Scrum-Prozess. Eine nähere Erläuterung der verschiedenen Testarten befindet sich in Kapitel 3.2.

Der neue Ablauf sieht drei verschiedene Testaktivitäten zu jeweils unterschiedlichen Zeitpunkten während eines Scrum-Projekts vor. Als Erstes der Entwicklertest, der schon früh bestimmte Probleme vorbeugt, da er parallel zur täglichen Arbeit eines jeden Entwicklers durchgeführt wird. Mit dem Inkrementtest wird die monatliche Arbeit eines Sprints mit automatischen Testmethoden überprüft. Zum Abschluss des gesamten Projekts wird mit einem Releasetest das vollständige Produkt überprüft und vorhandene Fehler für eine anschließende Kundenabnahme beseitigt.

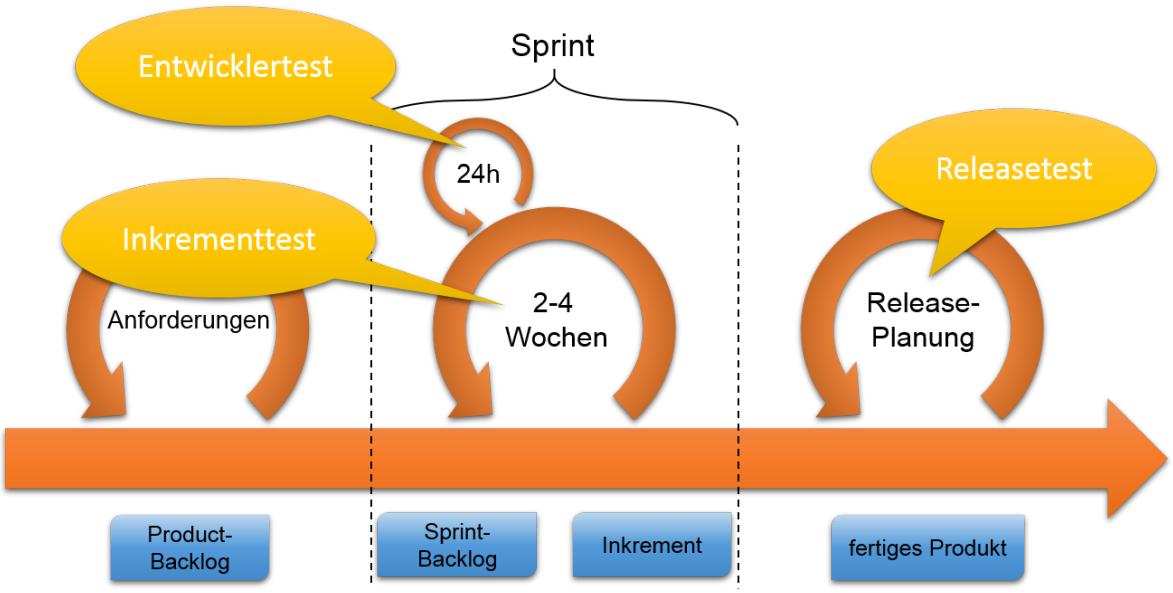

Abbildung 3.2: Eine mögliche Erweiterung des Scrum-Prozesses

3.2 Erklärung der verschiedenen Testarten

Im Folgenden sollen nun die verschiedenen Testarten, die den Scrum-Prozess erweitern können, näher vorgestellt und erklärt werden.

3.2.1 Entwicklertest

Die Arbeit jedes einzelnen Entwicklers während des Sprints wird durch einen täglichen Entwickler-Test überprüft und abgeschlossen. Dabei werden die erzielten Ergebnisse bezüglich ihrer Konformität mit den eigentlichen Sprint-Tasks überprüft. Jeder Entwickler ist für seine eigenen Testfälle verantwortlich, kann diese aber aus seinen Sprint-Tasks erstellen.

Durch (automatische) Unit-Tests und Continuous Integration werden die neu entstandenen Klassen und Methoden, sowie ihre Integration in bestehende Strukturen getestet.
Im Gegensatz zu [GG12] halten wir ein stündliches Durchlaufen der verschiedenen Tests für überflüssig, da ein Entwickler in den meisten Fällen in einer Stunde wohl kaum so viele neue Klassen entwickeln kann, dass diese gleich wieder getestet werden müssen. Ein täglicher Test sollte ausreichend sein.

3.2.2 Inkrementtest

Der Inkrementtest wird erst am Ende jedes Sprints durchgeführt und schließt somit die Entwicklung des neuen Inkrements ab. Da im Abschnitt zuvor bereits der tägliche Entwicklertest vorgestellt wurde, könnte man nun meinen, dass ein abschließender Sprint-Test eigentlich unnötig ist, da sämtliche Bestandteile schon im Vorfeld täglich überprüft wurden. Bei diesem Test soll der Schwerpunkt aber auch noch auf anderen Aspekten als die der konkreten Funktionsweise der einzelnen Module gelegt werden.

Es soll hauptsächlich das Zusammenspiel mit früher entwickelten Inkrementen getestet werden, also die Integration in das bisherige Produkt (Integrationstest). Gleichzeitig lässt sich auch sicherstellen, ob frühere Inkremente noch funktionieren oder in irgendeiner Art und Weise verändert wurden (Regressionstest). Als weitere Testarten können noch ein System-, ein Performanz- und ein Usabilitytest benutzt werden, mit welchen überprüft wird, ob das Inkrement als ein Ganzes funktioniert und vom Benutzer auch akzeptiert wird.

Im Wesentlichen werden im Inkrementtest also verschiedene Testarten miteinander vereint, die nicht die einzelnen Sprint-Tasks jedes Entwicklers verwenden, sondern das gesamte Sprint-Backlog.

Die Dauer und Intensität dieser Tests hängt davon ab, ob die zuvor in der Sprintplanung festgelegte „Definition of Done“ erfüllt wurde oder nicht. Diese Definition besagt, wann eine Aufgabe tatsächlich abgeschlossen ist und benötigt zur vollständigen Erfüllung auch alle nötigen Testaktivitäten. Am Ende eines Sprints liefert eine Auswertung Klarheit über den erreichten Stand.

3.2.3 Releasetest

Jedes Entwicklungsprojekt stellt mit einem abschließenden Test sicher, dass das entstandene Produkt fehlerfrei ist und vom Kunden auch akzeptiert wird. In Paper [GG12] wird vorgeschlagen, dass der Ablauf des Releasetests einem Scrum-Sprint ähnelt und sich somit über einen längeren Zeitraum erstreckt. Es finden täglich verschiedene Tests statt, welche zuvor in der Sprint-Planung festgelegt wurden. Auch hier findet sich wieder eine „Definition of Done“, die gegen Ende des Sprints bzw. des Releasetests ausgewertet wird. Nachdem diese letzte Testaktivität durchgeführt ist, kann die Abnahme durch den Kunden eingeleitet werden.

Da bei einem Scrum-Projekt verschiedene Teilprodukte bzw. Inkremente entstehen, macht es Sinn diese nun zusammen in einer fertigen Einheit zu überprüfen. Als Testbasis wird dafür nun gesamte Produkt-Backlog verwendet, das alle Anforderungen und Aufgaben enthält. Wie bereits erwähnt, ist der eigentliche Ablauf als Sprint organisiert. Somit wird auch wieder ein eigenes Sprint-Backlog benötigt wird. Dieses kann dann wiederum in tägliche Arbeitspakete aufgeteilt werden.

Mit Systemintegrationstests werden die einzelnen Bestandteile bezüglich ihrer Zusammenarbeit überprüft. Wichtig sind auch Akzeptanztests, mit denen verschiedene „Verhaltensweisen" der Software in bestimmten Situationen ausgewertet werden. Da hier Ergebnisse aus den einzelnen Sprints noch einmal begutachtet werden, können die bereits festgelegten Testfälle aus den Inkrementtests wieder verwendet werden, so dass kein zusätzlicher Aufwand entsteht.

Parallel zu den verschiedenen Tests findet das Debugging statt, um die gefundenen Fehler zu beseitigen. Ein am Ende des Tages erstellter Test-Report liefert eine Zusammenfassung und einen Überblick über die bis dahin erreichten Ziele.

3.3 Ergänzende Anpassungen

Damit der erweiterte Prozess von unserem Projektpartner umgesetzt werden kann, sind einige kleine Veränderungen notwendig. Diese betreffen einerseits die Einführung der neuen Tests, andererseits die bisherige Dokumentenstruktur. Um das Erstellen von Testfällen und die Überprüfung von diesen so zeitsparend wie möglich zu gestalten, sollten außerdem das Produkt-Backlog und das Sprint-Backlog erweitert werden.

Testziele Für jeden Sprint müssen Testziele formuliert werden. Was wird getestet? Wie lange? Wer ist dafür verantwortlich? Ein Sprint ist nur erfolgreich, wenn diese Ziele erreicht wurden. Die Planung hierfür kann in den Sprintmeetings stattfinden.

Produkt-Backlog Bereits bei der Formulierung der User-Stories muss auf testbare Formulierungen geachtet werden. Hier können Formulierungen aus jeweils unterschiedlichen Perspektiven hilfreich sein.

Sprint-Backlog Für das Sprint-Backlog werden wie bisher die User-Stories in Aufgaben aufgeteilt. Auch hier gilt, dass alle Formulierungen und Aussagen testbar sein müssen. Um eine Übersicht der Testaktivitäten zu erhalten, empfiehlt es sich, das Backlog um neue Spalten wie zum Beispiel „Getestet" oder „Testfälle" zu erweitern. Tabelle 3.1 zeigt hierfür ein mögliches Beispiel-Backlog.

ID	Name	Status	Beschreibung	Testfälle	Getestet	Gefundene Fehler	Priorität
...	...	...	...	...	...	...	...

Tabelle 3.1: Mögliche Erweiterung des Sprint-Backlogs für die Testplanung

3.4 Tool-Unterstützung für den Systemtest

Von unserem Projektpartner wurde gewünscht, den Systemtest automatisieren zu können. Eine Automatisierung sorgt dafür, dass wiederkehrende und auch teilweise zeitintensive Arbeiten vereinfacht werden und sich somit im Laufe der Zeit keine Nachlässigkeiten einschleichen. Die frei gewordenen Ressourcen können somit an anderen Stellen sinnvoller eingesetzt werden, ohne eine Verschlechterung der Qualität befürchten zu müssen.
Für diesen Arbeitsvorgang gibt es verschiedene Werkzeuge und Programme, die sich im Umfang und den vorhandenen Möglichkeiten sehr ähneln. Als Beispiele seien hier *Canoo WebTest*, *FitNesse* oder *Selenium* genannt. Wir haben uns für letzteres Tool entschieden, da es eine Vielzahl an Programmiersprachen unterstützt und als open-source Projekt verfügbar ist.

Selenium ist eine Open-Source-Zusammenstellung von Werkzeugen, die zum Testen von Webapplikationen verwendet werden können. Dabei steuert Selenium einen Webbrowser über JavaScript um Testfälle ausführen zu können. Es werden verschiedene Browser auf unterschiedlichen Plattformen unterstützt. Mit der integrierten Sprache „Selenese“ ist es möglich, Interaktionen mit einer Webanwendung aufnehmen zu lassen und diese Tests automatisiert beliebig oft zu wiederholen. Selenium eignet sich vor allem für Browser-Kompatibilitäts-Tests und funktionale Systemtests.

3.4.1 Selenium

Selenium setzt sich aus mehreren Komponenten zusammen. Je nach Anwendungsgebiet sind dabei eine oder mehrere Komponenten nützlich.

Selenium IDE

Mit der Selenium IDE ist es möglich, Testfälle direkt im Browser aufzuzeichnen. Die IDE ist dabei bisher allerdings nur als Firefox-Plugin verfügbar. Einmal installiert, ist es ohne weitere Vorkenntnisse möglich, Testfälle zu erstellen und diese wiederzugeben. Dabei erfolgt das Speichern in einem Testscript, das bearbeitet werden kann. Bei der Testfallerstellung werden nicht nur Aktionen des Testers aufgezeichnet, sondern es ist auch möglich, Elemente der Oberfläche zu überprüfen und zu validieren.
Die Befehle in Selenium werden als „Selenese“ bezeichnet. Ein Testscript setzt sich aus mehreren Selenese-Befehlen zusammen, welche auch manuell eingegeben werden können.
Über User-Extensions kann Selenium IDE erweitert werden. Eine bereits verfügbare Erweiterung ist die Möglichkeit Schleifen verwenden zu können.

Selenium Remote Control

Selenium RC oder Selenium 1 ist umfangreicher als Selenium IDE. Zuerst werden die Testscripte von Selenese in eine andere Programmiersprache übersetzt, beispielsweise Java. Dabei können die Tests aus der IDE weiterverwendet werden. Die Ausführung kann in verschiedenen Browsern erfolgen. Für den Testreport gibt es verschiedene Möglichkeiten: von der simplen Ausgabe auf der Konsole bis zur Integration in Werkzeugen wie JUnit ist alles möglich.
Ein großer Vorteil von Selenium RC besteht darin, dass Testfälle mit verschiedenen Eingabewerten ausgeführt werden können. Dafür sind aber einige Entwicklungskenntnisse in der jeweiligen Programmiersprache nötig. Für die eigentliche Verwendung ist dann nur noch separater RC-Server nötig.

Selenium WebDriver

Selenium WebDriver oder auch Selenium 2 ist die neueste Entwicklung dieses Toolkits. Es basiert auf Selenium RC, fügt aber einige neue Erweiterungen hinzu, verbessert das Interface und den Umgang mit dynamischen Seiten. Da sich WebDriver immer noch in Entwicklung befindet, ist die Sprachunterstützung noch nicht auf dem Stand von Selenium RC. Es ist aber möglich, Projekte aus Selenium RC weiter zu verwenden. Auch ein separater Server ist nun nicht mehr zwingend notwendig.

Selenium Grid

Mit Selenium Grid ist es möglich, die Tests parallel auf verschiedenen Maschinen durchzuführen. Zusätzlich unterstützt Selenium Grid die Testdurchführung in verschiedenen Browsern.

Für was kann Selenium verwendet werden?

Hauptsächlich lässt sich Selenium für Systemtests einsetzen. Die dabei testbaren Prüfkriterien sind funktionale und temporale Tests. Operationale Tests sind nicht möglich. Es ist nicht entscheidend, nach welchem Verfahren (Black-, White-, Grey-Box) die Testfälle erstellt wurden. Allerdings müssen sich sowohl Tester als auch Programmierer an die Schnittstellenbeschreibungen halten, da Selenium ansonsten die korrekten Elemente nicht ansprechen kann.

In welchen Browsern kann getestet werden?

Im Moment unterstützt Selenium WebDriver die nachfolgend aufgelisteten Browser. Dabei ist zu beachten, dass die Aufzeichnung und Wiedergabe der Testfälle weiterhin nur in Firefox möglich ist. Die Testausführung in den anderen Browsern erfolgt über Selenium WebDriver.

- Google Chrome
- Internet Explorer 6, 7, 8, 9, 10 (32- und 64-Bit)
- Firefox
- Safari
- Opera

Welche Programmiersprachen werden unterstützt?

Selenium RC kann eine Vielzahl an Programmiersprachen und Frameworks über eigene Treiber einbinden und verarbeiten.

- C# (verfügbare Frameworks: NUnit)
- Haskell
- Java (verfügbare Frameworks: JUnit, TestNG)
- JavaScript
- Objective-C
- Perl
- PHP
- Python (verfügbare Frameworks: unittest, pyunit, robot framework)
- R
- Ruby (verfügbare Frameworks: RSpec, Test::Unit)

3.4.2 Unsere Empfehlung

Um sich am Anfang an Selenium zu gewöhnen und einzuarbeiten, bietet sich der Einstieg über die IDE an. Damit lassen sich schnell erste Testfälle erstellen. Diese können dann nach und nach erweitert werden.
Für den professionellen Umgang würden wir Selenium WebDriver empfehlen. Es vereint die Stärken aus einem gleichnamigen, von Google-Mitarbeitern gegründeten Projekt zusammen mit Selenium RC. Durch die Einbindung des Betriebssystems, um auf den Browser zugreifen zu können, stehen weitere Testmöglichkeiten zur Verfügung.
Wenn die Testfälle in verschiedenen Browsern getestet werden sollen, bietet sich die Verwendung von Selenium Grid an, da dadurch viel Zeit gespart werden kann.
Die Unterstützung von mobilen Betriebssystemen wie Android und iOS erlauben auch in der Zukunft eine Qualitätssicherung der eigenen Produkte auf den passenden Endgeräten.

4 Fazit

Die bisherigen Abläufe bei unserem Partner für diese Prozessanalyse haben zwar funktioniert, an verschiedenen Stellen waren aber deutliche Verbesserungen nötig. Die Hauptprobleme bei der Dokumentation und beim Testen wurden von uns erkannt und von unserem Partner angenommen. Danach stand der eigentlichen Ausarbeitung in Form von Verbesserungsvorschlägen nichts mehr im Wege.
Wir haben eine Möglichkeit gefunden, wie ein regelmäßiges Testen, vor allem ein wiederholbarer Systemtest, möglich ist. Es sollte in Zukunft Entwicklertests, Inkrementtests und Systemtests geben, die zu unterschiedlichen Zeitpunkten auf Fehler hinweisen und ein frühes Eingreifen ermöglichen. Durch das vorgestellte Testtool werden die einzelnen Testarten unterstützt und vereinfacht.
Wir hoffen, dass unser Partner die Verbesserungsvorschläge langfristig einsetzen kann und sein Prozess dadurch nachhaltig verbessert wird.

Literaturverzeichnis

[GG12] S. Geisen, B. Güldali. Agiles Testen in Scrum – Testtypen und Abläufe. *OBJEKTspektrum (Online Themenspecials)*, (Agility/2012):1–4, 2012. (Zitiert auf den Seiten 15, 16 und 17)

[RWN] C. Rukes, W. Weich, D. Neuhaus. Analyse und Bewertung des Tools Selenium. URL http://winfwiki.wi-fom.de/index.php/Analyse_und_Bewertung_des_Tools_Selenium.

[sela] Selenium Blog mit Beispielen und Anwendungen. URL http://it-kosmopolit.de/Selenium/blog/selenium-blogs/selenium_blogs.php.

[selb] Selenium Documentation. URL http://docs.seleniumhq.org/docs/.

Alle URLs wurden zuletzt am 02.06.2014 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

(Verena Käfer, Florian Gänßlen, Burak Incearap)