Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Gregor Swatek

Optimierung agiler Softwareentwicklungsprozesse für Webentwicklung mit Hilfe von Agile Design

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Gregor Swatek

Optimierung agiler Softwareentwicklungsprozesse für Webentwicklung mit Hilfe von Agile Design

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Zhen Ru Dai
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 10. Februar 2012

Gregor Swatek

Thema der Arbeit
Optimierung agiler Softwareentwicklungsprozesse für Webentwicklung mit Hilfe von Agile Design

Stichworte
agile design, agile softwareentwicklung, scrum

Kurzzusammenfassung
Obwohl Scrum in den letzten Jahren erhebliche Erfolge vorzuweisen hat, haben sich in der Praxis diverse Probleme hervorgetan. Scrum sieht cross-funktionale Teams vor die alle erforderlichen Kenntnisse vorweisen um der bevorstehen Aufgabe gerecht zu werden. Dabei wird immer wieder festgestellt, dass gerade bei designspezifischen Aufgabe, der Designer sich nur schwer in den Prozess integrieren lässt. Um dieses zu verbessern und damit den Prozessfluss und die Wertschöpfungskette aufrechtzuerhalten, wird in dieser Arbeit eine Erweiterung des Scrum Prozesses, um weitere Artefakte und Etappen, vorgenommen. Der Designer gewinnt dadurch an Bedeutung im Scrum Team und wird als ein vollwertiges Mitglied anerkannt. Zugleich wird das Design der Anwendung nachhaltig und Flexibel durch inkrementelles Vorgehen aufgebaut. Das inkrementelle Design im Zusammenspiel mit den neuen Artefakten und Methoden wird als „Agiles Design“ bezeichnet.
Gregor Swatek

Title of the paper
Optimizing agile software development process for webdevelopment by agile design

Keywords
agile design, agile development, scrum

Abstract
Although Scrum has achieved considerable success in recent years, putting it into practice has revealed numerous problems. Scrum is typcially practicised by people with cross-functional skills who do the actual work. It is often noted that integrating design experts for design specific tasks is rather hard to accomplish. This thesis presents new artifacts and stages to improve and maintain the process flow and value chain. As a consequence designers become more important in the Scrum team and are recognised as full members and the design of the application built is more sustainable and flexible. This new approach that combines incremental design and the new artifacts and methods is called Agile Design.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation . . . 2
- 1.2. Zielsetzung und Problemstellung . . . 3
- 1.3. Aufbau der Arbeit . . . 4

2. Verwandte Arbeiten **5**
- 2.1. Überblick und Zusammenfassung bereits veröffentlichter Themen . . . 5
- 2.2. Vergleich der einzelnen Arbeiten in Bezug zur Thesis . . . 8

3. Grundlagen **10**
- 3.1. Klassische Vorgehensmodelle . . . 11
- 3.2. Was bedeutet Agilität? . . . 12
- 3.3. Agile Vorgehensmodelle im Allgemeinen . . . 13
- 3.4. Scrum im Detail . . . 16
  - 3.4.1. Rollen . . . 17
  - 3.4.2. Artefakte . . . 18
  - 3.4.3. Etappen . . . 20
  - 3.4.4. Zusammenfassung: Scrum Ablauf . . . 21
- 3.5. Softwareunternehmen vs. Werbeagenturen — Herkunft, Modelle und Kulturen 23
  - 3.5.1. Werbeagenturen . . . 23
  - 3.5.2. Softwareunternehmen . . . 24
- 3.6. Allgemeine Probleme und Herausforderungen . . . 26

4. Konzeption **29**
- 4.1. Rollenvergleich: Web-Entwickler vs. Designexperten . . . 30
  - 4.1.1. Designexperten . . . 30
  - 4.1.2. Web-Entwickler . . . 30
- 4.2. Konkrete Problemfelder und kritische Aspekte . . . 32
  - 4.2.1. Gemeinsamkeiten zwischen agilen und User-centered Designmethoden 32
  - 4.2.2. Unterschiede der beiden Methoden . . . 32
  - 4.2.3. Konkrete Probleme und deren Symptome . . . 33
  - 4.2.4. Tätigkeitsspezifischer Sprint . . . 35
  - 4.2.5. Kommunikationsdefizite während eines Sprints . . . 38
  - 4.2.6. Behinderungen im Sprint . . . 40
- 4.3. Mögliche Lösungsvorschläge . . . 41
  - 4.3.1. Kompetenzen des Designexperten . . . 42

4.3.2. Einführung des ‘Sprint-Zero’ . . . 43
4.3.3. Zusammenarbeit während des gesamten Sprints . . . 44
4.4. Zusammenfassung . . . 47
4.4.1. Optimierter Prozessverlauf . . . 47
4.4.2. Vorteile und Gefahren für Entwickler und Designexperten . . . 49
4.4.3. Die 14 Prinzipien des Agile Designs . . . 51

5. Realisierung **52**
5.1. Produktvision . . . 53
5.1.1. Was ist eine Produktvision? . . . 53
5.1.2. Warum braucht man eine Produktvision? . . . 54
5.1.3. Der Elevator Pitch . . . 55
5.2. Von der Produktvision zum Produkt . . . 57
5.2.1. Anwendung der AOF-Methode . . . 57
5.2.2. Epics und Stories - Aufbau und Unterschiede . . . 58
5.3. Interaktive UX-Pattern-Bibliothek . . . 61
5.3.1. Ursprung und Nutzen des interaktiven Styleguides . . . 61
5.3.2. HAML und SASS . . . 63
5.3.3. Aufbau und Architektur . . . 67
5.3.4. Meta-Konzepte im Styleguide . . . 70
5.4. Paperprototyping als Kommunikationsschnittstelle . . . 81
5.4.1. Sinnvolle Anwendung von Paper Prototyping . . . 82
5.4.2. Durchführung von Paperprototyping . . . 83
5.4.3. Vorteile und Nachteile von Paperprototyping . . . 85

6. Fallstudie **86**
6.1. Einleitung . . . 87
6.2. Fragebogen zur Fallstudie über „Agile Design“ . . . 87
6.3. Auswertung der Ergebnisse . . . 88
6.3.1. Kommunikation . . . 89
6.3.2. Zusammenarbeit . . . 89
6.3.3. Prozess . . . 90
6.3.4. Tools . . . 91

7. Zusammenfassung **93**
7.1. Fazit . . . 93
7.2. Ausblick . . . 95

A. Fragebogen **96**

B. Auswertung **103**

Literaturverzeichnis **128**

Glossar **133**

Tabellenverzeichnis

4.1. Konkrete Probleme deren Symptome und mögliche Lösungen Teil 1 von 3 . . 33
4.2. Konkrete Probleme deren Symptome und mögliche Lösungen Teil 2 von 3 . . 34
4.3. Konkrete Probleme deren Symptome und mögliche Lösungen Teil 3 von 3 . . 35
4.4. Praktiken zur Vermeidung der Parallelverarbeitung (Miller, 2005) 50

5.1. Beispiel für SASS Variablen 65
5.2. Beispiel für SASS Verschachtelung 65
5.3. Beispiel für SASS Mixins 66
5.4. Beispiel für SASS Selektoren Vererbung 67
5.5. Vorteile und Nachteile von Paperprototyping 85

Abbildungsverzeichnis

3.1. Das Wasserfallmodell 11
3.2. Das V-Modell 12
3.3. Agiler Software Entwicklungsprozess 15
3.4. Beispielgrafik für ein Burndown Chart in Scrum 19
3.5. Der Scrum Prozess 22
3.6. Gleichgewicht zwischen Vorhersage und Anpassung 26

4.1. Gängiger Erstellungsprozess für ein Design 37
4.2. Unterschiedliche Kommunikationsarten und deren Auswirkungen 39
4.3. Inkrementelles Design (Agile Design) 48

5.1. Positive Auswirkungen auf den Prozess durch inkrementelles Design 52
5.2. Aufbau und Architektur des Interaktiven Styleguides 69
5.3. Beispiel eines Standard Blueprint Grid-Layout 71
5.4. Aufbau eines Beispiel Layouts 72
5.5. Aufbau und Anordnung eines normalen Formulars 73
5.6. Formular Elemente mit nach links angeordneten Bezeichnungen 75
5.7. Formular Elemente mit nach rechts angeordneten Bezeichnungen 75

Listings

5.1. HTML5 Codebeispiel für den Vergleich zwischen HTML und HAML 63
5.2. HAML Codebeispiel für den Vergleich zwischen HTML und HAML 63
5.3. Beispiel für ein ERB-Template einer Ruby-on-Rails Anwendung 64
5.4. Beispiel für ein HAML-Template einer Ruby-on-Rails Anwendung 64
5.5. SASS Definition des Container Elementes (siehe auch Abb. 5.3) 71
5.6. SASS-Code zum Beispiel Layout (siehe Abb. 5.4) 72
5.7. HAML Beispiel für ein Standard Formular (siehe Abb. 5.5) 73
5.8. SASS Beispiel für ein Standard Formular (siehe Abb. 5.5) 74
5.9. Formular Erweiterung um Links ausgerichtete Feldbezeichner (siehe Abb. 5.6) 75
5.10. Formular Erweiterung um Rechts ausgerichtete Feldbezeichner (siehe Abb. 5.7) 75
5.11. HAML Beispiel für Buttons 76
5.12. SASS Definitionen für Buttons 76
5.13. SASS Definitionen für Farben 77
5.14. SASS Definitionen für Typographie-Mixins 78
5.15. SASS Definitionen für typographische Grundeinstellung 78
5.16. SASS Definitionen zum globalen zurücksetzen aller Standarddefinitionen der Browser 79
5.17. SASS Definitionen über das Verhalten von Hyperlinks 79

1. Einleitung

Im Februar 2001 wurde bei einem Treffen von 17 Experten in Utah der Begriff „agil“ geboren. Dieser ersetzte die bis dahin gebräuchliche Bezeichnung „leichtgewichtige Methoden“. Dies war die Geburt agiler Softwareentwicklung und somit auch von Scrum. Vier Jahre später hat Forrester Research eine Untersuchung zur agilen Softwareentwicklung veröffentlicht. Dabei wurde festgestellt, dass bereits 14 % der Unternehmen aus Nordamerika und Europa Projekte unter Zuhilfenahme von agilen Prozessen realisieren. Anfang 2010 hat Forrester Research unter dem Titel „Agile Softwareentwicklung ist Mainstream“ eine weitere Studie veröffentlicht. Das Ergebnis der Befragung ergab, dass nun 35 % der Teilnehmer agile Methoden in ihren Projekten einsetzen. Diese Zahlen beweisen, dass agile Softwareentwicklungsprozesse sich mittlerweile bei vielen Unternehmen etabliert haben und in keinem modernen Unternehmen mehr fehlen dürfen. Agile Softwareentwicklung ist jetzt gerade einmal zehn Jahre jung, trotzdem ist es an der Zeit, den Status quo zu betrachten. Scrum ist mittlerweile eines der populärsten agilen Vorgehensmodelle, aus diesem Grund wird Scrum in dieser Arbeit genauer betrachtet. Dabei fällt ein Punkt besonders ins Auge: Scrum sieht cross-funktionale Teams vor, welche alle Fähigkeiten besitzen, um die bevorstehende Aufgabe zu bewältigen. Die Rolle des Scrum-Teams und dessen Aufgabe im Projekt, sich auf Sprintziele zu einigen und Features auszuliefern, ist eines der Kernartefakte von Scrum. Des Weiteren sieht Scrum eine hohe Interaktion unter den einzelnen Teammitgliedern vor. Doch gerade dabei werden in Zusammenhang mit designspezifischen Aufgaben Konflikte in der Praxis sichtbar, die einen konstanten Prozessfluss verhindern und die es daher zu vermeiden oder zumindest zu verringern gilt.

1.1. Motivation

Scrum ist ein Vorgehensmodell der agilen Softwareentwicklung und setzt auf Werte wie Transparenz, Kommunikation und Feedback. Die Erfahrungen haben gezeigt, dass sich designspezifische Aufgaben nur schwer in einen agilen Entwicklungsprozess integrieren lassen. Der Designexperte ist meist ein Freiberuflicher, also ein externer dazugekaufter Mitarbeiter. Er arbeitet dabei parallel zum Entwicklerteam. Aufgrund der parallelen Zusammenarbeit wird der Kommunikationsfluss unterbrochen, die transparente Kommunikation ist aber essentieller Bestandteil von Scrum. Es entstehen also gewisse Konfliktpotentiale, die den Prozessfluss eines agilen Modells erheblich behindern. Dadurch verlängert sich die Projektlaufzeit, und somit steigen auch die Kosten und Risiken des Projektes. Die Herausforderung für die vorliegende Arbeit ist es, herauszufinden, warum sich ein Designexperte in der Realität nur bedingt in einen agilen Prozess integrieren lässt. Hierbei ist es besonders interessant, dass Scrum eigentlich ein Team vorsieht, welches alle nötigen Fähigkeiten besitzt, um der Aufgabe gerecht zu werden. Es stellt sich also die Frage, um welche Regeln oder Artefakte man Scrum in Bezug auf die Entwicklung von Produkten, die designspezifische Aufgaben beinhalten, erweitern müsste, damit ein reibungsloser Prozess gewährleistet wird. Diese Thesis befasst sich mit der Optimierung des Prozesses der agilen Softwareentwicklung, indem die Konfliktpotentiale analysiert werden und versucht wird, konkrete Lösungsansätze in Form von Regeln oder Tools in den Prozess zu integrieren und zu etablieren.

1.2. Zielsetzung und Problemstellung

Die Praxis hat gezeigt, dass durch externe Projektbeteiligte das Wissen und die Arbeitsabläufe sich orthogonal zueinander verhalten. Dies hat zur Folge, dass die Kommunikation erheblich beeinträchtigt wird. Je weiter das Projekt vorangeschritten ist, desto schwieriger wird es, dieses parallel gebildete Wissen zentral im Projektteam zu sammeln. Häufig hängt das damit zusammen, dass „das Design zum bestehenden Entwicklungsprozess einfach dazu gekauft wird — wie ein Möbelstück, das man am Ende in die neue Wohnung stellt." (Zitat: Wolfgang Wopperer) Es findet also weder Kommunikation statt noch fließt in irgendeiner Art und Weise Feedback ein, welches direkten Einfluss auf das Produkt nehmen könnte. Es ist wichtig, so früh wie möglich im Projekt mit dem Design zu beginnen und dabei die Entwickler in den Designprozess und die Designexperten in den Entwicklungsprozess zu integrieren. Aus diesem Grund wird das Design inkrementell in mehreren Schritten und zeitgleich zur Entwicklung der Anwendung erstellt und verbessert, genauso wie es bei agiler Softwareentwicklung üblich ist, zum Ende einer Iteration ein lauffähiges Stück Software auszuliefern. Das Ziel dieser Arbeit ist es, dieses inkrementelle Design — auch bekannt als „Agile Design" — unter Zuhilfenahme der in dieser Arbeit vorgestellten Artefakte und Etappen in den Scrum-Prozess zu integrieren. Im besten Fall lässt sich mit Hilfe dieser Arbeit ein flüssiger Agiler Prozess etablieren, der weniger Unterbrechungen und parallele Arbeit, dafür aber mehr Kommunikation und Zusammenarbeit beinhaltet.

1.3. Aufbau der Arbeit

Die vorliegende Arbeit ist in folgende Kapitel unterteilt:

→ In **Kapitel 2** werden Veröffentlichungen zu diesem Thema zusammengefasst aufgeführt und Vergleiche zu der vorliegenden Arbeit gezogen.

→ **Kapitel 3** befasst sich mit den Grundlagen der Softwareentwicklungsprozesse. Es werden klassische und agile Methoden erklärt und miteinander verglichen. Scrum wird hier im Detail dargestellt. Zusätzlich gibt es einen kurzen Ausblick auf die allgemeinen Probleme und Herausforderungen des agilen Designs.

→ **Kapitel 4** besteht hauptsächlich aus analytischen Informationen über Agenturen und Softwareunternehmen zur Identifizierung der Unterschiede in der Kultur und Arbeitsweise. Darauf aufbauend werden konkrete Problemfelder aus Kapitel 3 betrachtet, und im weiteren Verlauf wird näher auf mögliche Lösungswege eingegangen.

→ **Kapitel 5** ist das Praxiskapitel. Hier werden Konzepte und Realisierungsvorschläge erarbeitet. Es wird deutlich gemacht, durch welche Methoden man den erwähnten Problemen bei der Integration des agilen Designs vorbeugen kann.

→ **Kapitel 6** beinhaltet eine Fallstudie zum agilen Design. Um die konkrete Prozess- und Schnittstellenproblematik zu beleuchten, wurden mehrere Personen aus dem agilen Umfeld zum agilen Design befragt. Dieses Kapitel beinhaltet den Fragebogen und die Auswertungen der Resultate.

→ In **Kapitel 7** werden die gesamte Problematik und die Lösungen zusammengefasst dargestellt und bewertet. Außerdem wird ein Ausblick auf mögliche Themen gegeben, die es noch zu betrachten gilt.

2. Verwandte Arbeiten

Dieses Kapitel befasst sich mit bereits veröffentlichen Arbeiten, die sich ebenfalls mit dem inkrementellen Design und dessen Integration in einen agilen Prozess beschäftigen. Im ersten Abschnitt werden detaillierte Zusammenfassungen dieser Veröffentlichungen aufgeführt. Aus den einzelnen Arbeiten werden die Informationen festgehalten, die von höchster Relevanz für den weiteren Verlauf der eigenen Untersuchung sind. Der darauf folgende Abschnitt beinhaltet eine relativ allgemeine Analyse der in den einzelnen Arbeiten behandelten Fakten. Aus ihr werden Verbindungen und Unterschiede zu dieser Arbeit gezogen und aufgezeigt.

2.1. Überblick und Zusammenfassung bereits veröffentlichter Themen

Jennifer Ferreira, James Noble und Robert Biddle beschreiben in ihrer Arbeit vier verschiedene Projekte. Deren Versuch ist es, durch die Integration der UI-Designer ein agiles cross-funktionales Team aufzubauen. Dabei lassen sich einige Thesen ableiten:

- Aus dem Vorweggestalten ergeben sich Vorteile.
- Es sollte möglichst viel vorweg gestaltet werden.
- Der wesentliche Teil eines Interaktionsdesigns besteht aus der Anwenderforschung.
- Das Design lernt aus der Implementierung der Entwickler.
- Sowohl das Usability-Testing als auch die Implementierung haben Einfluss aufeinander.
- Agilität verändert die Beziehung zwischen Designexperten und Entwicklern.

Der Artikel beschreibt leider nur sehr kurz und flüchtig den eigentlichen Prozess. Auf derselben Grundlagentheorie wie bei Ferreira et al. untersuchen die Autoren David Fox, Jonathan Sillito und Frank Maurer ebenfalls die Integration von User-centered Design (im weiteren Verlauf dieser Arbeit als UCD abgekürzt) mit agilen Methoden. Sie versuchen, die

existierenden Modelle in ein möglichst generelles Modell zu erweitern. Es wurden Interviews mit zehn Teilnehmern durchgeführt, jeweils mit einem Mitglied eines größeren Teams; die Mehrheit der Teilnehmer waren Designexperten, nur ein paar waren Entwickler. Als Grundlage für den Prozess wird das Diagramm mit zwei parallelen Spuren von Desirée Sy einbezogen. Aus diesem Artikel gehen drei Ansätze hervor — der generische Ansatz, der spezialisierte Ansatz und der Hybridansatz. Im generischen Ansatz gibt es keinen explizit ausgewiesenen Designspezialisten im Team, die Rolle wird von einem Entwickler übernommen. Im spezialisierten Ansatz hingegen besitzt das Team einen Design- und Usability-Spezialisten. Der Hybridansatz hingegen beinhaltet eine Person, die beide Rollen vereint, es ist ein Designspezialist, der ebenfalls Erfahrung aus der Softwareentwicklung mitbringt (Fox u. a., 2008, S. 63–72). Jim Ungar, der selbst ein Interaktionsdesigner ist, beschreibt Agile UCD als Design, das in einer aus agilen Methodiken bestehenden Hülle ausgeführt wird. Weiterführend bezeichnet er Agile UCD als eine Art Wissenschaft, bei der der Informationsfluss zum Team in der gleichen Art und Weise erfolgen sollte wie bei der Softwareentwicklung. Ungar führt in seinem Artikel die Vorteile eines Designstudios auf. Hierbei handelt es sich um einen kollaborativen Workshop im Agile-UCD-Prozess. „Das Designstudio ist ein schneller Prozess, der es Designexperten, Entwicklern und Kunden ermöglicht, gemeinsam verschiedene Designalternativen auszuarbeiten und zu untersuchen. Teilnehmer erweitern ihre Fähigkeiten, indem sie ihre Standpunkte mit anderen austauschen und offen über die Stärken und Schwächen ihrer Arbeit diskutieren. Das erzeugte Design wird durch das Feedback aus diesem Workshop bereichert“ (Ungar, 2008, S. 520).

Das Designstudio setzt sich aus folgenden vier Hauptkomponenten zusammen:

- Benutzerrecherche (das Studio wird mit Informationen aus der Recherche beliefert)
- Design (viele schnell erzeugte Designs und Ideen)
- Studio (ein Tages-Workshop, um Alternativen zu evaluieren und Entscheidungen zu treffen)
- Teilnehmer (Designer und Nichtdesigner, die dazulernen möchten)

Der Artikel von Zahid Hussain, Wolfgang Slany und Andreas Holzinger untersucht den aktuellen Stand bei der Umsetzung von agilen Designmethoden in der Praxis. Die benötigten Informationen wurden durch Interviews eingeholt, welche mit Teilnehmern durchgeführt wurden, die bereits mit der integrierten Methodik des agilen Designs Erfahrung gesammelt haben. Weitere Daten wurden bei der Beobachtung der Teilnehmer in ihrem Arbeitsumfeld

gewonnen. Es wurden Teilnehmer aus unterschiedlichen Bereichen befragt, wie Usability-Experten, Entwickler, Programmmanager, Agile Coaches und Projektmanager. Aus der Studie geht hervor, dass Mitglieder agiler Entwicklungsteams immer mehr die Wichtigkeit der Integration dieser beiden Themen realisieren. Nahezu alle Teilnehmer haben erwähnt, dass die Integration der beiden Methoden die Qualität und Usability ihres Produktes gesteigert hat. Einige Teilnehmer betonten, dass diese Integration ebenfalls neue Werte im Team und dem Prozess erzeugt hat. Aus der Schlussfolgerung geht hervor, dass etwas Designarbeit im Voraus Verständnis für die Benutzer, deren Wünsche und die Produktvision schafft. Sowohl genaue als auch weniger genaue Prototypen wurden dafür eingesetzt (Hussain u. a., 2009, S. 279–289). Die Fallstudie von Maryam Najafi und Len Toyoshiba zeigt die Integration von agilen Designmethoden anhand von zwei Projekten. Im ersten Projekt, genannt „The Falcon Project“, konnte die Integration von Design und Entwicklung erfolgreich durchgeführt werden. Hierbei wurde darauf Wert gelegt, dass die beteiligten Personen regelmäßig an den Meetings teilnahmen. Alle Mitglieder des Teams waren in der gleichen Stadt angesiedelt. Bei diesem Projekt wurde der sogenannte Sprint-Zero eingeführt, indem die Anforderungen analysiert und die initialen User Stories erstellt wurden. Die durchgeführten Benutzertests hatten unmittelbare Auswirkung auf die Priorisierung der Produkt-Backlogs.
Das zweite Projekte, genannt „The Razor Project“, war aufgrund der räumlichen Trennung der Teilnehmer und der damit verbundenen fehlenden Kommunikation deutlich weniger erfolgreich als das erstere. Nachdem das Designteam mit dem Produktmanager einen Informationsaustausch über mehrere Runden betrieben hatte, wurden sehr detaillierte Prototypen erstellt. Während der Implementierung wurde das Designteam nicht in den Prozess einbezogen, so dass keinerlei Benutzertests und Recherchen erfolgten (Najafi u. Toyoshiba, 2008, S. 531–536).

Im Prozess wurden unter anderem folgende Kernpunkte identifiziert:

- Die Produktentwicklung fand in Isolation statt.
- Der Prozess hatte Wasserfallcharakter.
- Eine vollständige Feature-Implementierung resultiert nicht automatisch in einem fertigen Produkt.
- Durch fehlende Kommunikation wurde das Design missverstanden und nicht korrekt umgesetzt.

In ihrem Artikel zeigt Desirée Sy, wie in ihrem Unternehmen die Integration von Design und Entwicklung in einem agilen Kontext erfolgreich durchgeführt wurde. Sie beschreibt das

iterative Vorgehen am Design als „Zyklus“; dabei wird ebenfalls auf ein Sprint-Zero eingegangen, bei dem die initial benötigten Informationen gesammelt werden, um die Benutzer und das Endprodukt besser zu verstehen.
Es wird ein Prozess vorgestellt, welcher zwei parallel verlaufende Spuren besitzt, eine für die Designexperten und eine für die Entwicklung. Bei der Durchführung eines jeden Zyklus koordinieren die beiden Parteien ihre Arbeit dadurch, dass das Designteam immer genau einen Zyklus vor der Implementierung im Voraus arbeitet. Somit wird sichergestellt, dass genügend Zeit bleibt, die Anforderungen und das Design für den kommenden Zyklus vorzubereiten und dabei das im vorherigen Zyklus umgesetzte Design zu testen und aufgedeckte Fehler zu beseitigen. Sy macht darauf aufmerksam, dass es sich hierbei zwar um parallel laufende Pfade handelt, diese sollen allerdings simultan ablaufen. Sie beschreibt, dass in der Praxis oftmals die Entwicklung eines Produktes zu einem Zeitpunkt beginnt, bei dem die Anforderungen noch nicht geklärt sind. Dies resultiert dann darin, dass Features umgesetzt werden, für die es noch kein fertiges oder überhaupt kein Design gibt. Sy bezeichnet ihre Lösung als „Just-in-time design“ und beschreibt damit, dass im agilen Kontext einzelne Features zu einer bestimmten Zeit fertiggestellt werden. Somit ist das Designteam auch nicht gezwungen, das gesamte Design vorweg zu gestalten. Des Weiteren erläutert Sy in ihrem Artikel wie mit komplexen Designs umgegangen wurde. Es war offensichtlich, dass diese von Natur aus eine längere Zeit in Anspruch nehmen würden. Aus diesem Grund wurden große Designs in mehrere kleine Design-Chunks aufgeteilt. Diese Design-Chunks bauen aufeinander auf und können aufgrund ihrer Größe besser integriert werden, indem sie inkrementell zum gesamten Design hinzugefügt werden (Sy, 2007, S. 112–130).

2.2. Vergleich der einzelnen Arbeiten in Bezug zur Thesis

Alle Arbeiten, die für diesen Vergleich genauer betrachtet wurden, haben ebenso wie die vorliegende Thesis einen hohen Praxisbezug. Aus fast allen Arbeiten ging hervor, dass der Sprint-Zero ein wesentlicher Aspekt bei der Integration von UCD und der agilen Entwicklung ist. Fox beschreibt, dass beim Sprint-Zero lediglich Informationen zwischen dem Designexperten und dem Kunden ohne Beteiligung der Entwickler ausgetauscht wurden. Aus den Erfahrungen, die für diese Thesis gesammelt wurden, ging deutlich hervor, dass bei einem solchen initialen Sprint möglichst alle Beteiligten teilnehmen sollten. Ferreira erläutert in seiner Studie, dass man in einem Sprint-Zero versuchen sollte, möglichst viel vorweg zu gestalten. Das widerspricht dem Ansatz von Sy und der vorliegenden Thesis. Sy bestätigt mit ihrer Arbeit, dass ein frühes Design möglichst aus groben und fundamentalen Design-Chunks

und Prototypen bestehen sollte, damit das Design durch ständige Einarbeitung des Feedbacks verbessert werden kann.
Nachdem initial Arbeit ins Design geflossen ist, beschreiben die Studien von Sy und Fox, dass der weitere Ablauf parallel, aber zeitgleich stattgefunden hat. Dieser Ansatz konnte auch in der vorliegenden Thesis bestätigt werden. Dabei ist es jedoch wichtig, dass die parallelen Pfade zeitgleich abgearbeitet werden, womit sich kurze und effektive Feedback-Wege ergeben, durch die das Design inkrementell erweitert werden kann. In den meisten Fällen werden dafür wöchentliche Meetings vereinbart, hier wird jedoch der Ansatz verfolgt, möglichst oft während eines Sprints das nötige Feedback zu erhalten. Sy beschreibt leider nicht konkret genug, wie die Parallelitäten möglichst gering gehalten werden und wann genau das Feedback erfolgen sollte. Sie empfiehlt, im ersten Sprint nach dem Sprint-Zero durch hohe Entwicklungskosten Zeit für das Design zu gewinnen. Die Praxis hat jedoch gezeigt, dass dies nicht immer möglich ist. Es kann durchaus vorkommen, dass am Anfang eines Projektes durch geringen Designaufwand, wie z. B. das Erzeugen eines Basislayouts oder grundlegender Elemente, genug Design für die erste Entwicklungsphase geschaffen werden können, ohne dass hohe Entwicklungskosten entstehen müssen.

Durch erfolgreiche Integration der Designexperten und Entwickler im agilen Prozess ist ein gemeinsames Meeting wichtiger Bestandteil der Zusammenarbeit. Während dieses Meetings haben die beteiligten Projektteilnehmer die Möglichkeit, ihre Meinungen und Gesichtspunkte auszutauschen und offen darüber zu diskutieren. Diese Erkenntnis wird ebenfalls durch das Designstudio von Unger bestätigt. Es beinhaltet viele gute Aspekte, jedoch müsste der Prozess optimiert werden. Unger spricht davon, dass im Designstudio das ‘beste’ Design gefunden werden soll, dabei läuft man jedoch Gefahr, in einen ‘Design by committee’-Prozess zu geraten. Bei dem in dieser Arbeit vorgestellten Konzeptionsmeeting werden Meinungen und Ideen offen diskutiert und ausgetauscht, jedoch hat der Designexperte und vor allem der Kunde das letzte Wort, die anderen liefern lediglich technischen Input.
Die kontinuierliche Zusammenarbeit während der Produktentwicklung, vor allem auf technischer Ebene, wird im Allgemeinen als ein sehr wichtiger Aspekt erkannt. „Fertige Designs oder Prototypen einfach umzusetzen führt zwangsläufig zu Missverständnissen und Fehlinterpretation.“ (Zitat: Najafi zitiert nach (Najafi u. Toyoshiba, 2008)) Najafi macht durch seinen Vergleich zweier Projekte besonders deutlich, dass eine enge Zusammenarbeit und das gemeinsame Verständnis unverzichtbar dabei sind, agile Designmethodiken und die agile Entwicklung optimal miteinander zu kombinieren. Aus keiner der Studien geht hervor, wie man die Kommunikation ungeachtet der gemeinsamen Zusammenarbeit noch weiter ausbauen kann.

3. Grundlagen

In diesem Kapitel werden die grundlegenden Vorgehensmodelle aufgezeigt. Es gibt einen kleinen Überblick über klassische und agile Vorgehensmodelle. Als Beispiele für klassische Vorgehensmodelle werden das Wasserfall- und das V-Modell herangezogen. Außerdem wird auf agile Vorgehensmodelle im Allgemeinen eingegangen und dabei das Scrum-Vorgehensmodell im Detail abgehandelt. Scrum ist eines der Vorgehensmodelle, welche sich in den letzten Jahren etabliert haben, und wird am häufigsten im Unternehmenskontext eingesetzt. Aus diesem Grund wird Scrum als Beispiel für agile Vorgehensmodelle näher betrachtet. Die unterschiedlichen Unternehmenskulturen der Softwareentwickler und Designexperten werden im weiteren Verlauf verglichen. Anschließend werden allgemeine Probleme, die von den unterschiedlichen Kulturen herrühren, aufgezeigt.

3.1. Klassische Vorgehensmodelle

In der Welt der klassischen Vorgehensmodelle erfolgt eine Abbildung des Software-Lebenszyklus auf eine definierte Abfolge von unterschiedlichen Phasen.

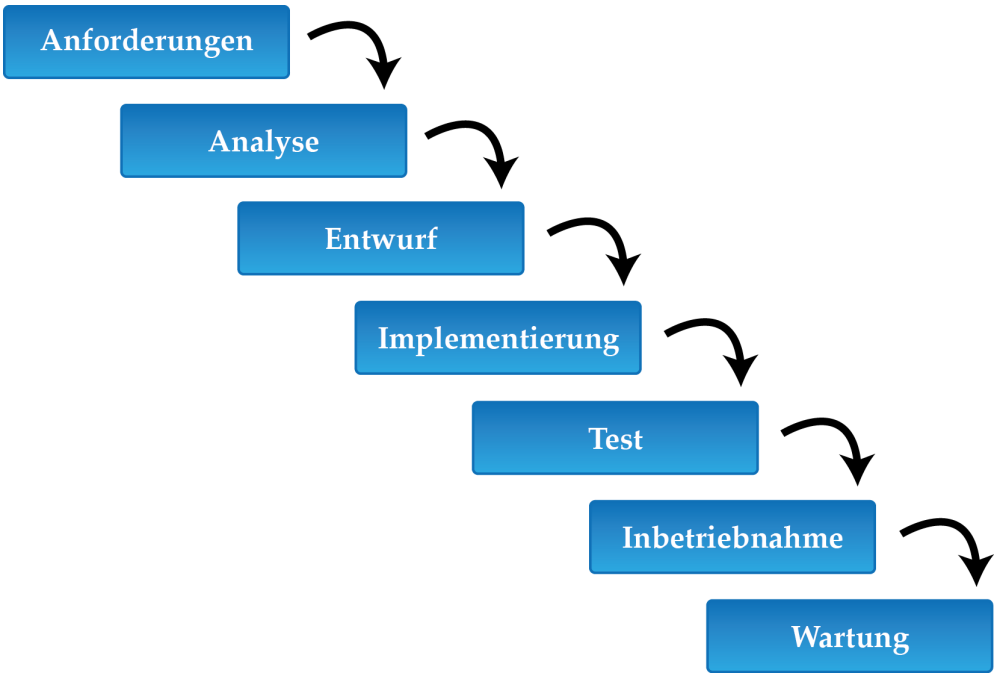


Abbildung 3.1.: Das Wasserfallmodell

Sobald sich Anforderungen an das Produkt ändern, müssen diese in einem komplett neuen Zyklus durchlaufen werden. Das aber ist mit hohem Aufwand verbunden und somit sehr zeit- und kostenintensiv. Diese Art von Vorgehensmodell ist sehr starr und bietet nur sehr wenig Flexibilität, selbst beim Übergang der einzelnen Phasen. Das Wasserfallmodell berücksichtigt nur teilweise die fehlende Möglichkeit von Rückschritten, deshalb ist es nur möglich, von einer Arbeitsphase zu der direkt davor liegenden Phase zurückzukehren (Zuser u. a., 2001, S. 45ff).
Auf der Grundlage des Wasserfallmodells wurde dann später das V-Modell entwickelt. Grundsätzlich ist das V-Modell nicht ausschließlich auf Softwareprojekte anwendbar, sondern kann auch für andere Projekte übernommen werden. Es entstand ursprünglich aus der Rüstungsindustrie, allerdings wird es heute nicht mehr weiterentwickelt. Abb. 3.1 zeigt in welche sechs Phasen sich das V-Modell gliedert.

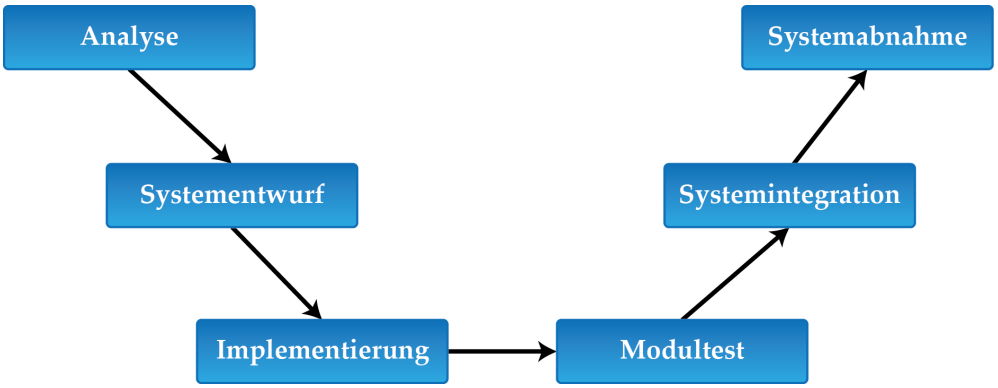


Abbildung 3.2.: Das V-Modell

Seine sechs Phasen sind in V-Form angeordnet und werden wie folgt durchlaufen: Die Modultest-Phase überprüft das Artefakt der Phase 3, welches sich um den Feinentwurf und die Implementierung kümmert. Die Phase 5 (Systemintegration) überprüft wiederum die Ergebnisse des Systementwurfs in Phase 2. Am Ende bei der Systemabnahme werden dann schließlich die Ergebnisse der Systemanalyse aus der ersten Phase konsultiert. Sobald in einem dieser Schritte Widersprüche auftreten, muss zur überprüften Phase zurückgekehrt werden. Dieser Rückschritt funktioniert nun vom gescheiterten Modultest zurück zur Implementierung wahrscheinlich noch recht akzeptabel. Wenn aber bei der Systemabnahmeüberprüfung festgestellt wird, dass Anforderungen nicht ausreichend verstanden wurden, führt dies in den meisten Fällen zum Projektabbruch. Die zeitlichen Abstände zwischen der ersten und letzten Phase liegen bei größeren Projekten durchaus auch einmal bei mehreren Jahren. Daraus resultiert, dass man sowohl beim Wasserfall- als auch beim V-Modell nur schlecht und verbunden mit hohen Kosten auf Änderungen der Anforderung oder anderer äußerer Umstände reagieren kann (Zuser u. a., 2001, S. 47).

3.2. Was bedeutet Agilität?

Bevor man nun mit der Betrachtung der agilen Softwareentwicklung beginnt, sollte man zuerst konkreter auf die Bedeutung des Wortes „Agilität“ eingehen, um ein besseres Verständnis für die agile Softwareentwicklung zu bekommen. Man geht von einer Welt aus, die einem ständigen Wandel unterliegt und sich im Hinblick auf folgende Aspekte verändern könnte:

- Änderung im Projektteam (z. B. Teambesetzung)

- Änderung im technologischen Umfeld (z. B. Wegfall der Unterstützung für bestimmte verwendete Technologien)
- Änderung der Anforderungen durch den Kunden
- Änderung durch Konkurrenz (z. B. Veröffentlichung einer neuen Version eines Konkurrenzproduktes)
- Änderung im Markt

Sowohl klassische als auch agile Vorgehensmodelle bieten Möglichkeiten und Werkzeuge an, auf Veränderungen zu reagieren. So existiert in der klassischen Softwareentwicklung z. B. das Change Management und das Risikomanagement. Der agile Vertreter Extreme Programming nimmt lediglich eine kurzfristige Planung vor, um auf Änderungen in den Anforderungen reagieren zu können. Hierbei ist ein wesentlicher Unterschied zu erkennen. Während Veränderungen bei der klassischen Vorgehensweise als ein erfolgreich zu verwaltendes Risiko betrachtet werden, sehen die agilen Modelle die Änderungen als eine Chance. Somit fordert die klassische Softwareentwicklung, diese Änderungen möglichst zu vermeiden bzw. die Auswirkungen auf das Projekt so gering wie möglich zu halten. Die agile Softwareentwicklung wiederum akzeptiert diese Veränderungen und versteht sie als integralen Bestandteil eines Projektes (Stein, 2004).
Agilität zeichnet sich aber nicht nur durch die Akzeptanz der Veränderung aus. Agilität zeichnet sich vor allem dadurch aus, dass sie Veränderungen einfordert, wenn es nötig ist. Dies ist besonders dann vorteilhaft, wenn man eine Veränderung auslöst, der die Konkurrenz nicht oder nur mit großem Aufwand folgen kann. Eine Veränderung der Anforderungen ist dann sinnvoll, wenn sich durch die geänderten Anforderungen und die realisierte Software der Geschäftswert erhöht. Zwei Bedingungen müssen demnach erfüllt sein, damit ein Unternehmen als agil bezeichnet werden kann (Highsmith, 2002, S. 5f und S. XXIII):

- Das Unternehmen akzeptiert und bewältigt Wandel.
- Das Unternehmen nutzt und löst Wandel zum eigenen Vorteil aus.

3.3. Agile Vorgehensmodelle im Allgemeinen

Im Februar 2001 haben Mitglieder der Agile Alliance ihre Vorstellungen von agiler Softwareentwicklung als das sogenannte „Agile Manifest“ niedergeschrieben. Die Grundprinzipien des agilen Manifests lauten wie folgt: (?)

- **Individuen und Interaktionen** gelten mehr als Prozesse und Werkzeuge.
- Eine **funktionierende Software** gilt mehr als eine umfassende Dokumentation.
- Die **Zusammenarbeit mit dem Kunden** steht über den Verträgen.
- Das **Reagieren auf Veränderung** ist wichtiger als das Befolgen eines Plans.

Agile Vorgehensmodelle haben also als wichtigstes Ziel, im gesamten Prozess funktionierende Software für den Kunden auszuliefern. Dabei soll die Integration sich ständig verändernder Anforderungen an die Software durch ein iteratives Vorgehen ermöglicht werden. Anforderungen definieren, was das Produkt können sollte. Dabei dienen die Geschäftsziele als Grundlage für die Anforderungen. Das Problem ist nun aber, dass die Geschäftsziele zum Anfang der Entwicklung meist nicht klar definiert sind und dem Entwickler nicht klar kommuniziert werden. Es findet also ein doppelter Übersetzungsprozess statt: von den Geschäftszielen zu den Anforderungen und von den Anforderungen zum Produkt.
Diese beiden Übersetzungsprozesse stellen die größten Herausforderungen in der Softwareentwicklung dar, denn es ist weder trivial, für ein Geschäftsziel die exakten Anforderungen zu definieren, noch, gegebene Anforderungen in ein softwaretechnisches Artefakt zu überführen. Die Anforderungen lassen sich noch leicht mit dem bestehenden Produkt vergleichen, indem das Produkt getestet wird. Wie können aber die Geschäftsziele an den gegebenen Anforderungen überprüft werden? Dies ist nur mit einem Produkteinsatz möglich, denn nur mit ihm zeigt sich, ob die Geschäftsziele mit dem gegebenen Produkt erreicht werden können. Agile Softwareentwicklung macht also den frühen Einsatz des Produktes zum Grundpfeiler des Projekterfolges (it agile, 2011). Um nun möglichst früh einen Release des Produktes zu erzeugen, werden die Geschäftsziele möglichst explizit aufgestellt, gewichtet und priorisiert. Darauf aufbauend werden nun für die am höchsten priorisierten Geschäftsziele feingranulare Anforderungen erstellt und für einen ersten Release geplant und entwickelt. Durch automatische Tests wird immer sichergestellt, dass das Produkt den Anforderungen entspricht und funktionsfähig ist. Nachdem der erste Release zum Einsatz kommt, hat der Kunde die Möglichkeit, seine Erfahrungen in die Geschäftsziele einfließen zu lassen und evtl. die Priorisierung dieser Ziele zu verändern. Des Weiteren hat er nun auch die Option, neue Ziele zu definieren. Diese Abfolge wird iterativ im zyklischen Ablauf durchgeführt (Wolf, 2007).

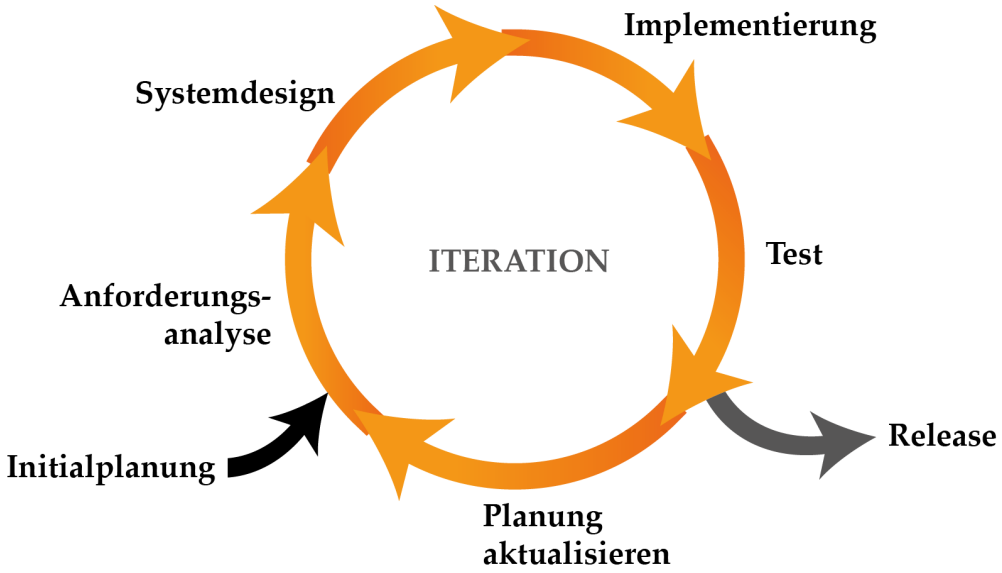


Abbildung 3.3.: Agiler Software Entwicklungsprozess ohne Bezug zu einem bestimmten Modell.(Vogt, 2007)

Werte der agilen Vorgehensmodelle (Wolf, 2007, S. 10ff)

- Alle Projektbeteiligten müssen viel miteinander kommunizieren, um ein gemeinsames Ziel zu verfolgen und gemeinsam über Entscheidungen, Probleme und gewählte Lösungswege zu sprechen.
- Gerade weil Software von Natur aus komplex ist, soll diese Schwierigkeit durch die agilen Methoden verringert werden, damit die Software möglichst wenig Komplexität auf technischer, organisatorischer und methodischer Ebene besitzt.
- Feedback wird in agile Methoden eingeführt, damit Zwischenergebnisse bewertet werden können und überprüft werden kann, ob der angestrebte Geschäftszweck erfüllt wurde.
- Mut wird dort benötigt, wo die agilen Methoden auf Widerstand stoßen. Es müssen Entscheidungen getroffen werden und eine offene, direkte und ehrliche Kommunikation muss geführt werden.
- Respekt ist ausschlaggebend dafür, dass die Projektbeteiligten offen und ehrlich und angstfrei miteinander kommunizieren. Respekt ist wichtig, um alle im Team produktiv einzubinden und keine Außenseiter zu schaffen.

Die größten Vorteile aus diesem iterativen Prozess sind die folgenden:

- Transparenz im Projekt
- Erstellen von Prognosen für den weiteren Verlauf sehr leicht
- Flexibilität der Anforderungen durch Priorisierung
- Schneller Systemeinsatz durch Implementierung der für das Produkt wichtigsten Anforderungen
- Schnelles erreichen eines Geschäftswert
- Direkte Kommunikation
- Kontinuierliches Feedback
- Gegenseitige Vertrauen zwischen den Projektbeteiligten

3.4. Scrum im Detail

In diesem Kapitel wird das Scrum-Framework näher betrachtet. Scrum ist iterativ und beruht auf der Ansicht, dass Projekte oft zu umfangreich sind, um durchgängig planbar zu sein. Es versucht durch das Zusammenspiel von bestimmten Rollen, Artefakten und Etappen die Komplexität zu verringern. Man wird in die Lage versetzt, „produktiv und kreativ Produkte mit dem höchstmöglichen Wert auszuliefern“ (Schwaber u. Sutherland, 2011). Scrum ist ein leichtgewichtiges agiles Vorgehensmodell, welches zwar einfach zu verstehen, aber nur schwer zu meistern ist.
Scrum ist ein Framework, bei dem unterschiedliche Prozesse und Techniken zum Einsatz kommen, um die Wirksamkeit des Produktmanagements und Vorgehens sichtbar zu machen. Auf dieser Grundlage können dann Verbesserungen eingeleitet werden. Der Prozess beinhaltet verschiedene Komponenten aus definierten Rollen, Artefakten und Etappen. „Jede Komponente des Frameworks dient einem spezifischen Zweck und ist wesentlich für die erfolgreiche Anwendung von Scrum“ (Schwaber u. Sutherland, 2011).

Das Scrum-Framework basiert auf der empirischer Prozesssteuerung[1]. Diese stützt sich auf drei Säulen (Schwaber u. Sutherland, 2011):

- Transparenz
- Überprüfung
- Anpassung

Transparenz meint, dass für diejenigen Beteiligten, die für die Ergebnisse verantwortlich sind, die wesentlichen Aspekte des Prozesses erkennbar werden. Überprüfung bedeutet, dass Anwender die Artefakte und Fortschritte, die auf dem Weg zum Ziel gemacht werden, überprüfen müssen, um Anomalien zu entdecken. Sollte dabei festgestellt werden, dass diese Abweichung außerhalb akzeptabler Grenzen liegt und somit auch das Produktergebnis nicht zu akzeptieren sein wird, müssen Anpassungen des Prozesses oder des Arbeitsgegenstandes vorgenommen werden, um weitere Abweichungen zu minimieren.

3.4.1. Rollen

Das Scrum-Team definiert drei Rollen: Product Owner, das Entwicklungsteam und der Scrum Master die in diesem Abschnitt näher erläutert werden.

Product Owner
Der Product Owner ist Herr über das Product Backlog und stets für den wirtschaftlichen Erfolg des Projektes verantwortlich. Dieser Aspekt muss für ihn bei allen Entscheidungen immer im Vordergrund stehen. Infolgedessen erstellt und priorisiert der Product Owner die Anforderungen im Product Backlog. Hierbei ist es wichtig, dass er die Bedürfnisse seiner Kunden kennt und diese auch gut in das Team kommunizieren kann. Außerdem spielt die enge Zusammenarbeit und eine möglichst geringe räumliche Trennung vom Team eine besondere Rolle. Am Ende eines Sprints ist es seine Aufgabe, bei einem Review Meeting die implementierten Features abzunehmen.

[1] Empirismus basiert auf der Grundannahme, dass Wissen aus Erfahrung gewonnen wird und dass Entscheidungen auf Grundlage bekannter Fakten getroffen werden

Scrum Master
Ein Scrum Master übt eine Führungsrolle aus, ohne über Weisungsbefugnis zu verfügen. Er übernimmt also nicht die klassische Rolle des Projektmanagers, er ist eher ein Moderator und sorgt dafür, dass das Team effektiv arbeiten kann und den Geschäftswert aufrechterhält. In seiner Rolle als Moderator beseitigt er Hindernisse im Sprint, die das Team bei der effektiven Entwicklung am Produkt hindern.

Team
Scrum-Teams organisieren sich selbst und sind cross-funktional aufgestellt. Cross-funktionale Teams verfügen über alle Fähigkeiten, die benötigt werden, um die gewünschten Ergebnisse zu erreichen, ohne dabei von anderen Personen abhängig zu sein. Durch das iterative Ausliefern von Teilprodukten und deren inkrementelle Erweiterung werden die Feedback-Gelegenheiten maximiert. Das gängige Scrum-Team-Modell hat die Herausforderung, die Flexibilität, Kreativität und Produktivität des Teams stetig zu verbessern. Das Entwicklungsteam hat zur Aufgabe, zum Ende eines Sprints potentiell auslieferbare und fertige Produkt-Artefakte zu erstellen. Die Verantwortung für das Arbeitsergebnis am Ende eines jeden Sprints liegt somit beim gesamten Team.

3.4.2. Artefakte

Artefakte repräsentieren die erbrachte Arbeit auf unterschiedliche Art und Weise. Sie bieten die nötige Transparenz und schaffen Gelegenheit zur Überprüfung und Anpassung der Arbeit. Diese Artefakte sind speziell dafür gedacht, die Transparenz zu maximieren und die wichtigsten Informationen aufzuzeigen. Sie stellen sicher, dass das Team ein fertiges Teilprodukt erfolgreich ausliefern kann.

Product Backlog
Das Product Backlog ist eine nach dem Geschäftswert geordnete Liste von Features, Anforderungen, Verbesserungen und Fehlerbehebungen, welche die Änderungen an dem Produkt beeinflussen. Die alleinige Verantwortung für das Product Backlog inklusive dessen Inhalte und Reihenfolge trägt der Product Owner. Es ist niemals vollständig, sondern entwickelt sich permanent weiter, damit es an die Anforderungen im Wettbewerb und die Nützlichkeit angepasst werden kann. Das Backlog ist so aufgebaut, dass die hoch priorisierten Einträge detailliert beschrieben sind, damit sie in den vorgegebenen Timeboxen fertiggestellt werden können. Ein gutes Product Backlog wird nach unten hin ungenauer und verliert an Detailtiefe (Schwaber u. Sutherland, 2011).

Sprint Backlog

Ein Sprint Backlog enthält alle Elemente aus dem Product Backlog, die nötig sind, um das nächste lauffähige Stück Software zu liefern. Es ist die Prognose des Teams darüber, welche Funktionalität im nächsten Release enthalten sein wird und was nötig ist, um diese Funktionalität zu erreichen. Der Product Owner und das Team einigen sich beide auf das im Sprint Backlog festgehaltene Ziel. Das Sprint Backlog macht somit die Schritte transparent, die das Team für notwendig erachtet, um das Ziel zu erreichen. Es bildet die Grundlage für spätere Abnahmen beim Sprint Review (Schwaber u. Sutherland, 2011).

Burndown Chart

Das Burndown Chart ist ein weiteres Scrum-Artefakt, welches zur Transparenz während eines Projektes beiträgt. Es bietet die Sicht auf den Restaufwand eines jeden Sprints. Im Idealfall fällt die Kurve kontinuierlich — daher Burndown — und der Restaufwand ist am Ende des Sprints null. Das Chart lässt bereits während des Sprints erkennen, ob der anfangs geschätzte Aufwand umgesetzt werden kann. Es ist ein Werkzeug, welches relativ einfach, aber deutlich aufzeigt, wie sich die Projektlaufzeit während der Entwicklung verändert. Somit kann man die gewonnenen Informationen in die Planung einbeziehen.

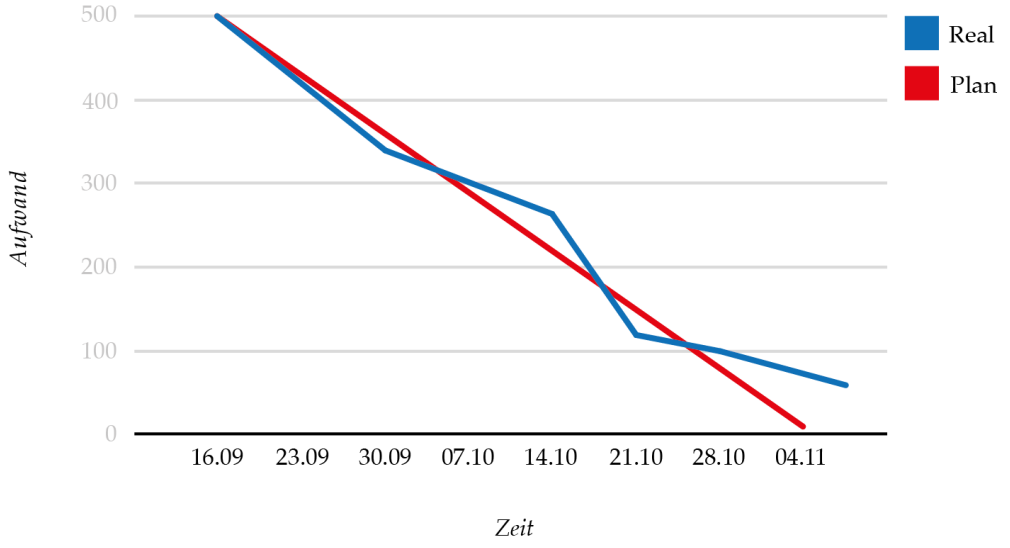


Abbildung 3.4.: Beispielgrafik für ein Burndown Chart in Scrum

Impediment Backlog

Das Impediment Backlog listet alle Umstände des Projektes auf, die das Team daran hindern, effizient das Sprintziel zu erreichen. Das Team hat die Verantwortung, während des Daily Scrum diese Hindernisse zu benennen. Weitere Hindernisse können ebenfalls bei der

Retrospektive am Ende eines Sprints aufgedeckt werden. Der Scrum Master ist dann dafür zuständig, diese Hindernisse gemeinsam mit dem Team schnellstmöglich auszuräumen (Scherer, 2011).

3.4.3. Etappen

Bei Scrum kommen fest definierte Etappen zum Einsatz, die möglichst eine Regelmäßigkeit herstellen und somit die Notwendigkeit für außerordentliche Besprechungen minimieren. Hierbei werden Timeboxen verwendet, die jedes Ereignis durch eine vorgegebene maximal Dauer beschränken, um eine Verschwendung von Zeit für Planungszwecke zu minimieren. Die Ereignisse sollen für eine möglichst große Transparenz sorgen, damit eine optimale Überprüfung und Anpassung stattfinden kann (Schwaber u. Sutherland, 2011).

Sprint Planning
Beim Sprint Planning wird die Arbeit, die für den kommenden Sprint zu erledigen ist, geplant. Am Planning ist das gesamte Scrum Team beteiligt. Zu Beginn des Meetings erklärt der Product Owner die für den kommenden Sprint priorisierten Backlog-Einträge. Sollten zu diesem Zeitpunkt nicht geschätzte Features darunter sein müssen diese Backlog-Einträge zuerst vom Team geschätzt werden. Daraufhin werden einzelne Features bzw. User Stories in das Sprint Backlog gelegt. Am Ende einigt sich das gesamte Team auf das geplante Sprintziel.

Sprint
Der Sprint ist das zentrale Element von Scrum. Die Dauer eines Sprints ist abhängig von der Teamgröße und schwankt zwischen ein und vier Wochen. Während eines Sprints wird potentiell auslieferbare Software erstellt. Hierfür zieht das Team die am höchsten priorisierten Einträge aus dem Sprint Backlog heran. Während des Sprints arbeitet das Team störungsfrei an den Features. Störungsfrei bedeutet u. a., dass keine neuen Features von außen dazwischengeschoben werden können. Am Ende eines Sprints steht immer ein lauffähiges, getestetes und inkrementell verbessertes Produkt.

Daily Scrum
Das Daily Scrum Meeting ist ein auf eine Timebox von 15 Minuten beschränktes kurzes Meeting, bei dem sich das Team synchronisiert und einen Plan für die kommenden 24 Stunden erstellt. Das geschieht durch Beantwortung von drei Fragen:

- Was habe ich bis jetzt gemacht?
- Was mache ich heute?

- Was hat mich bei meiner Aufgabe behindert?

Das Meeting wird jeden Tag zur gleichen Zeit am gleichen Ort durchgeführt, um die Komplexität zu verringern. Dabei werden keine Diskussionen oder Reportings ausgetragen, es dient ausschließlich dem Informationsaustausch des Teams untereinander.

Sprint Review
Am Ende eines jeden Sprints trifft sich das Scrum-Team zum Sprint Review, damit das bis dahin fertiggestellte Produktartefakt untersucht werden kann. Dabei präsentiert das Team die Ergebnisse und der Product Owner prüft die erreichten Ziele. Auf dieser Grundlage — und anhand eventueller Änderungen am Product Backlog während des Sprints — stellt das Scrum-Team die nächsten Schritte für den kommenden Sprint zusammen. Das Resultat aus diesem Meeting ist ein überarbeitetes Product Backlog, welches die potentiellen Elemente für den kommenden Sprint enthält (Schwaber u. Sutherland, 2011).

Sprint Retrospektive
In der Retrospektive geht es darum, dass das Scrum-Team sich selbst reflektiert und einen Plan zur Verbesserung erstellt, der dann im nächsten Sprint umgesetzt werden soll. Die Retrospektive findet nach dem Review statt und ist ebenfalls mit einer Timebox versehen. Bei diesem Meeting werden von jedem Teilnehmer folgende drei Fragen beantwortet:

- Was ist gut gelaufen?
- Was ist schlecht gelaufen?
- Was kann ich verbessern?

Hierfür schreiben die Teilnehmer auf Karteikarten, was sie während des letzten Sprints verrückt, traurig oder froh gemacht haben. Die Karten werden vom Scrum Master an eine Wand angebracht und in Cluster unterteilt. Jeder Teilnehmer darf max. 6 Punkte auf die Cluster verteilen und somit die Bereiche priorisieren, bei denen seiner Meinung nach Diskussionsbedarf besteht. Nach der Vergabe dieser Punkte wird dann über das Thema mit den meisten Punkten diskutiert, dabei moderiert der Scrum Master diese Diskussion und identifiziert die Kernpunkte. Am Ende der Diskussion werden die Verbesserungsvorschläge festgehalten und dokumentiert.

3.4.4. Zusammenfassung: Scrum Ablauf

Scrum beinhaltet drei elementare Rollen — Product Owner, Scrum Master und das Team. Der Product Owner ist Herr über das Product Backlog, er priorisiert dieses Backlog nach dem

Geschäftswert. In einem Sprint Planning werden die am höchsten priorisierten Features durch das Team in das Sprint Backlog überführt. Das Sprint Backlog beinhaltet somit alle Features, die für die nächste Iteration beauftragt werden. Ein Sprint ist in der Regel ein bis zwei Wochen lang. Während eines Sprints arbeitet das Team ohne Störungen von außen an den für diesen Sprint vorgesehenen Features. Des Weiteren werden täglich Daily Scrums abgehalten, bei denen sich das Team untereinander synchronisiert. Der Scrum Master moderiert alle anstehenden Meetings und unterstützt das Team dabei, störungsfrei zu arbeiten, indem er die Behinderungen schnellstmöglich aus dem Weg räumt. Am Ende eines jeden Sprints präsentiert das Team die Ergebnisse in einem Sprint Review, und der Product Owner nimmt sie ab und prüft, ob das Sprintziel erreicht wurde. Die Sprint Retrospektive am Ende eines Sprints, an der alle beteiligt sind, dient der Überprüfung des gesamten Prozesses. Bei diesem Treffen kristallisieren sich die Dinge heraus, die gut liefen, und diejenigen, die sich noch weiter verbessern lassen.

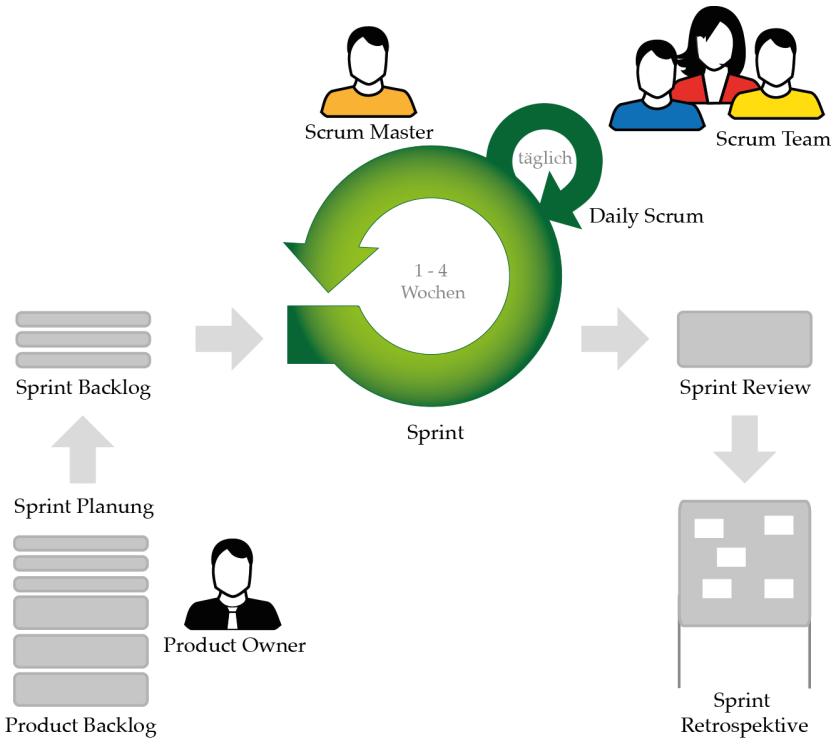


Abbildung 3.5.: Der Scrum Prozess

3.5. Softwareunternehmen vs. Werbeagenturen — Herkunft, Modelle und Kulturen

Um das bestehende Problem besser verstehen zu können, gilt es, zunächst die beiden unterschiedlichen Welten zu betrachten, die hier im agilen Prozess aufeinanderstoßen. Man sollte sich über die Herkunft, die unterschiedlichen Modelle und Kulturen im Klaren sein und sie verstehen.

3.5.1. Werbeagenturen

Der Hauptgrund, weshalb Grafikdesigner eine so unterschiedliche Ansicht und Denkweise haben, ist wohl der, dass sehr viele Agenturen ihre Herkunft in der Werbeindustrie haben und kampagnenorientiert sind. Der Kunde definiert ein abstraktes Ziel, darauf hin pitchen die „Kreativen“ darum, welche Idee umgesetzt wird, um dieses Ziel zu erreichen. Nach der Entscheidung für eine Idee folgen Grobkonzeption, Feinkonzeption, Entwürfe/Wireframes/Text, Ausarbeitung, Überarbeitung, Korrekturen, Druckvorlagen und dann die fertige Werbung.

Der Grafikdesigner ist es von früher her gewohnt, ein Bild oder Plakat zu gestalten, bei dem er sich sicher ist, das es nach dem Druck immer noch genauso aussieht wie auf dem Bildschirm. Es ist völlig egal, ob es ein Plakat, Werbebanner oder eine Illustration in einer Zeitschrift wird, es wird sich äußerlich und in seinem Verhalten nicht ändern. Damit ist die Arbeit des Designexperten abgeschlossen und er kann sich neuen Aufgaben widmen. Mit neuen Webtechnologien und vor allem den unterschiedlichen Browsern ist die Arbeit mit einer statischen Photoshop-Datei hingegen noch lange nicht getan. Internetseiten sind dynamische Produkte, sie können ihr Verhalten verändern, und die gestalterischen Aspekte verändern sich mit dem Inhalt der Seite.

Des Weiteren herrscht in einer Agentur eine andere Kultur. Es müssen Deadlines gehalten werden, d. h., wenn irgendetwas brennt oder erledigt werden muss, wird durchgearbeitet, bis es fertig ist — ein ganz normaler Alltag in einer Agentur. Dass die Qualität und Produktivität erheblich darunter leidet, wird sehr oft außer Acht gelassen. Als Ursache des Ganzen sind hier die Festpreisprojekte zu sehen, die dazu verleiten, möglichst viel in kurzer Zeit zu schaffen und dabei die Qualität zu vernachlässigen, um den größtmöglichen Profit herauszuschlagen. Als Resultat weiß oft hinterher keiner mehr so genau, warum etwas gebaut wurde oder von wem es überhaupt gebaut wurde. In der Agenturwelt herrschen zugleich ein starkes Hierarchiegefälle und zu lange und umständliche Kommunikationswege. Während einer

Kampagne wird jeder Schritt „nach oben“ abgestimmt und jede Ausarbeitung „nach unten“ delegiert. Es herrschen klare Hierarchien und eine strenge Arbeitsteilung, entsprechend gibt es viele Übergabepunkte, die erst einmal überwunden werden müssen.

Der Projektmanager spricht mit dem Kunden, koordiniert dessen Wünsche und teilt diese dem Team mit. Dies geschieht aber wiederum über den Director, dem die einzelnen Teammitglieder unterstellt sind. Somit ist das Team, welches die eigentlichen Aufgaben bewältigen soll, quasi ferngesteuert und vom relevanten Kommunikationskanal abgeschnitten. Nachdem der Projektmanager mit dem Kunden gesprochen hat, wird ein kleines Konzept erstellt und gleich angefangen zu gestalten. Danach geht es in das Templating, im Anschluss daran wird sofort entwickelt. Diese Arbeitsschritte haben meistens einen fließenden Übergang, ohne dass es eine Feedbackschleife zum Kunden gibt. Sollten Fragen auftauchen, haben diese einen längeren Weg, auf dem dann evtl. Informationen verloren gehen, was die Kommunikation erheblich beeinträchtigt. Meetings sind in Agenturen nicht besonders beliebt, man glaubt, dass oft zu viele Personen aus höheren Positionen zusammensitzen, wenig Produktives dabei herauskommt und es am Ende viel zu teuer war. Es scheint einfacher, Aufgaben zu delegieren und diese vom „Fußvolk“ erledigen zu lassen.

3.5.2. Softwareunternehmen

Auf der anderen Seite herrschte lange Zeit die Vorstellung, Software müsse „fabrikmäßig“ entwickelt werden — schließlich sei die Informatik eine Ingenieursdisziplin, und moderne Produkte stellt man auch in einem wiederholbaren, fehlerminimierten und hochautomatisierten Prozess her. Der Fehler dieser Betrachtungsweise liegt darin, dass man den falschen Abschnitt eines Produktentwicklungs- und Produktionsprozesses als Analogie benutzt. Produktion ist industrielle Vervielfältigung, die Produktentwicklung findet aber bereits vorher statt und funktioniert ganz anders. Produktentwicklung ist ein explorativer, feedbackintensiver und iterativer Prozess. An dessen Beginn wird ein Ziel formuliert, wobei ganz unterschiedliche Rollen wie Ingenieure, Designexperten und Tester beteiligt sind. Folglich entsteht aus Ideen, Möglichkeiten und Beschränkungen das Produkt. Erst ganz am Ende, wenn das Produkt quasi „fertiggestellt“ ist, wird es spezifiziert und an die Produktion zur Vervielfältigung übergeben. Die Vervielfältigung von digitalen Produkten ist aber eine schlichte bitweise Kopie. Entsprechend besteht ein Softwareentwicklungsprozess im Grunde ausschließlich aus einer ständigen Produktentwicklungsphase. Der Begriff „Entwickler“ findet hiermit auch wieder zu seiner ursprünglichen Bedeutung zurück (Basiert auf Gesprächen mit Wolfgang Wopperer, mindmatters GmbH & Co. KG, 2011).

Die Softwareentwicklung befindet sich bereits in einem großen Wandel. In der Vergangenheit gab es bereits die oben beschriebenen Strukturen. Die meisten Softwareprojekte waren Festpreisprojekte, was dazu führte, dass Entwickler bis tief in die Nacht Software bauten und dabei Codes produzierten, die später kein anderer Entwickler mehr lesen konnte außer sie selbst. Die Softwareentwicklung hat diese Probleme erkannt, und mit Einführung agiler Vorgehensmodelle dominiert diese Art des Vorgehens nicht mehr. Moderne und agile Unternehmen versuchen möglichst flache Hierarchien zu etablieren. Entwickler sind nicht mehr durch Projektleiter oder andere Instanzen ferngesteuert. Sie kommunizieren persönlich mit dem Kunden und tauschen sich mit diesem durch mehrere Feedbackschleifen aus. Gerade diese Direktkommunikation ist entscheidend, schließlich sind sie es, die die Implementierung der Kundenwünsche vornehmen.

Seit Einführung der agilen Vorgehensmodelle organisieren sich Teams selbst, sie besprechen mit dem Kunden, welche Teile des Produktes umgesetzt werden, und treffen eigenverantwortlich Entscheidungen, wie diese Teile implementiert werden sollen. Genau aus diesem Grund können sich die Entwickler viel besser mit dem Produkt identifizieren, was zu einer höheren Motivation und zur Verbesserung der Qualität führt, denn jeder Entwickler ist selbst für seine Implementierung verantwortlich. Die agilen Vorgehensmodelle sind noch recht jung und nur schwer in die festen Strukturen von Agenturen integrierbar. Die meisten Softwaredienstleister haben diese Entwicklung bereits hinter sich und haben den Vorteil der Agilität für sich entdeckt. Agenturen hingegen kommen nur schleppend damit voran, die Prozesse sind auf Print und Werbung ausgelegt, wo man kaum iterativ vorgehen kann, weil es ansonsten ungeheuer teuer zu werden droht. Bei der Entwicklung von Webanwendungen und bei der Arbeit im Web im Allgemeinen muss man aber umdenken. Man muss anpassungsfähig bleiben und auf Veränderungen am Markt und die entsprechenden Anforderungen möglichst schnell reagieren können.

3.6. Allgemeine Probleme und Herausforderungen

Die allgemeinen Herausforderungen bei der Integration eines Designexperten und der Anwendung des inkrementellen Designs im agilen Kontext besteht darin, dass Designexperten es gewohnt sind, im Vergleich zum Projekt insgesamt und den an ihm beteiligten Kräften weit im Voraus zu arbeiten und die Entwickler mit fertig gestalteten Oberflächen zu beliefern. In der Softwareentwicklung ist das Codefragment oder die Funktion, die ein Entwickler geschrieben hat, nie wirklich fertig, sie muss ständig verbessert und angepasst werden. Ebenso verhält es sich mit dem Design einer Webanwendung, das zwangsläufig aufgrund neuer Anforderungen oder des Feedbacks aus den Usability-Tests ebenso gepflegt und Schritt für Schritt verbessert und weiterentwickelt werden muss.
Scrum-Teams haben jedoch erkannt, dass es unmöglich ist, alle Designentscheidungen im Vorfeld zu treffen. „Versucht man nämlich im Vorfeld eine genaue Analyse durchzuführen und das Design zu erstellen, versucht man die Bedürfnisse der Anwender vorauszusehen“ (Cohn, 2010). Diese Vorgehensweise ist niemals ohne Fehler, und aus diesen Fehlern oder geänderten Anforderungen resultieren dann neue Anpassungen, die relativ teuer werden können. „Verzichtet man allerdings auf Analyse und Design und beginnt ohne Voraussicht unmittelbar mit der Implementierung, versucht man sich an die Bedürfnisse der Benutzer anzupassen“ (Cohn, 2010). Es ist also erforderlich, ein Gleichgewicht zwischen Vorhersage und Anpassung zu finden.

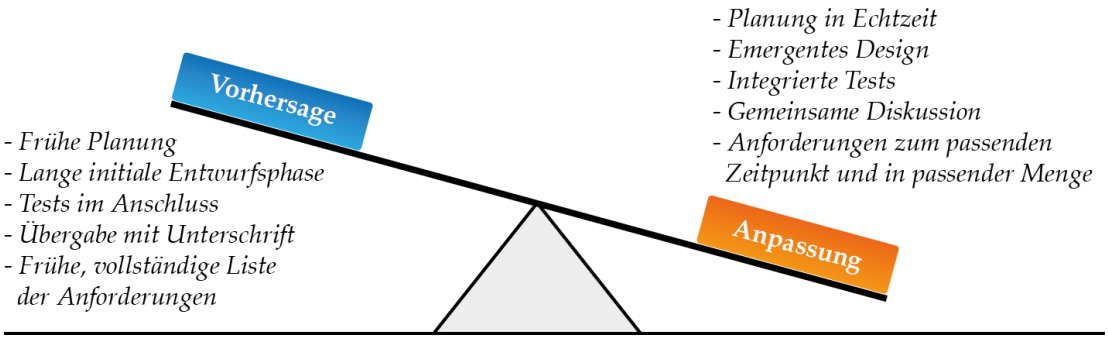


Abbildung 3.6.: Diese Grafik von Mike Cohn zeigt, dass ein Gleichgewicht zwischen Vorhersage und Anpassung nötig ist.(Cohn, 2010, S. 197)

Schon 1990 sagte der Redner und Autor Do-While Jones: „Ich bin nicht dagegen, für die Zukunft zu planen. Eine zukünftige Ausweitung der Fähigkeiten sollte durchaus in Betracht gezogen werden. Aber wenn der gesamte Designprozess stecken bleibt, weil versucht wird, zukünftige Anforderungen zu erfüllen, die vielleicht niemals Wirklichkeit werden, dann ist es an der Zeit, innezuhalten und sich zu überlegen, ob es nicht eine einfachere Möglichkeit gibt,

das aktuelle Problem zu lösen“[2] Die Herausforderung liegt also darin, solch ein Stillstand durch inkrementelles Design zu vermeiden. In der Softwareentwicklung hat man mittlerweile gelernt, sich nicht Sorgen über alle möglichen Anforderungen zu machen, sondern sie aus dem aktuell Gegebenen abzuleiten und sich daran anzupassen. Diese Mentalität muss auch in das agile Design übertragen werden, hierfür muss man „lernen, ohne umfassendes Design auszukommen“ (Cohn, 2010, S. 197).

Laut Mike Cohn ergeben sich bei diesem Wandel unter anderem folgende neuen Umstände, die mit der vorliegenden Thesis bestätigt werden können (Cohn, 2010, S. 197f):

- Die Planung wird schwieriger.
- Es wird schwieriger, die Arbeit auf mehrere Teams oder Einzelpersonen aufzuteilen.
- Es fühlt sich merkwürdig an, kein fertiges Design zu haben.
- Die Umarbeitungen sind unvermeidlich.

Scrum schreibt cross-funktionale Teams vor, um effektiv am Produkt arbeiten zu können. Auf dieser Grundlage muss der Designexperte als Teil des Teams verstanden und akzeptiert werden. Ein weiterer Aspekt dabei ist, dass selbst dann, wenn der Designexperte voll in das Scrum-Team integriert ist, nicht automatisch ein gemeinsames Verständnis geschaffen wird, welches zur korrekten Integration führt. Dieses erreicht man nur durch effektive Kommunikation und Zusammenarbeit in Kombination mit viel Disziplin. Bei folgenden Kommunikationsschnittstellen konnten Schwächen im Informationsaustausch identifiziert werden:

- Kunden und Designexperten
- Entwickler und Designexperten

[2] Jones Artikel „The Breakfast Cooker“ ist eine klassische Parabel für das, was schiefgehen kann, wenn Softwareentwickler das Design einer Lösung übertreiben, url: http://www.ridgecrest.ca.us/~do_while/toaster.htm

Scrum sieht eine enge Zusammenarbeit mit dem Kunden vor und liefert dafür bestimmte Werkzeuge, die dem Team dabei helfen, den Kunden regelmäßig in das Projekt zu integrieren und Feedback zu erhalten. Am besten wird Wissen persönlich vermittelt; besonders Details lassen sich bei dieser Face-to-Face Kommunikationsform — die oft vernachlässigt wird — am besten besprechen. Menschen geben eher bei einem persönlichen Gespräch Informationen preis, außerdem ist diese Art der Kommunikation reichhaltiger und liefert mehr Wert für das Produkt. Missverständnisse, Unklarheiten und offene Fragen lassen sich sofort beseitigen oder durch einfache Nachfragen aufklären (Cohn, 2010).

4. Konzeption

Das vorliegende Kapitel beschäftigt sich detaillierter mit den in Abschnitt 3.6 bereits erwähnten Themen, indem die allgemeinen Probleme und Herausforderungen als Grundlage genommen werden, um konkrete Konfliktpotentiale zu identifizieren. In Abschnitt 3.5 wurden Designagenturen und Software-Dienstleistungsunternehmen auf die Herkunft der unterschiedlichen Modelle und Kulturen, die in diesem Umfeld herrschen, untersucht. Damit die grundlegende Problematik bewusst gemacht werden kann, muss man zunächst die unterschiedlichen Arbeits- und Sichtweisen der Designexperten und Entwickler gegenüberstellen. Darauf aufbauend werden im weiteren Verlauf dieses Kapitels die Problemfelder und kritischen Punkte, die Auswirkungen auf die Zusammenarbeit, die Integration und den Prozess haben, genauer untersucht. Zum Schluss werden mögliche Lösungsvorschläge aufgeführt, die sowohl die Kommunikation als auch den Prozess an sich optimieren sollen. Diese Erkenntnisse dienen dann als Grundlage für Kapitel 5.

4.1. Rollenvergleich: Web-Entwickler vs. Designexperten

Auf die Herkunft und die Kulturen der beiden unterschiedlichen Welten wurde bereits in Abschnitt 3.5 konkret Bezug genommen. Es muss nun ein Schritt weiter gegangen werden, um die beiden unterschiedlichen Perspektiven besser verstehen zu können. Entsprechend werden die beiden Hauptakteure dieser Problematik untersucht und gegenübergestellt.

4.1.1. Designexperten

Der Designexperte besitzt meist einen Abschluss im Interaktions-, Kommunikations- oder Grafikdesign. Seine Werkzeuge sind meistens Papier und Stift oder Photoshop. Der Designexperte kennt sich gut mit Oberflächendesign und Designmustern aus, er weiß, wie Interface-Elemente miteinander interagieren müssen. Er verbringt gerne Zeit mit Benutzern seiner Arbeit, um zu recherchieren, ob und wie er die Usability seiner Arbeit weiter verbessern kann.
Ein Designexperte muss nun in einem agilen Prozess der Unterstützer sein, er muss mit seinem Spezialwissen dem Team dabei helfen, die Aufgabe zu lösen. Er sollte zusammen mit dem Team Gespräche führen, einen Wissensaustausch betreiben und dabei helfen, die Wünsche, Verhaltensweisen und Standpunkte der Benutzer dem Team verständlich zu machen. Seine wichtigste Aufgabe muss er darin sehen, alles zu tun, um die Teile des Produktes für den aktuellen Sprint zu liefern. Außerdem hat er die Aufgabe, nach vorne zu schauen und zusammen mit dem Product Owner den bevorstehenden Sprint im Auge zu haben.
Bei der Zusammenarbeit mit Entwicklern ist er dafür verantwortlich, ihnen das eigentliche Produkt hinter der technologischen Fassade zu verdeutlichen. Dafür unterstützt er sie beim Brainstorming oder nimmt an Konzeptionsmeetings teil, bei denen er zusammen mit dem Team Designlösungen austauscht und ihnen seine Designentscheidungen erklärt.

4.1.2. Web-Entwickler

Der Entwickler hat einen Abschluss in Computerwissenschaften, z. B. der Angewandten Informatik. Werkzeuge eines Entwicklers sind meist Programmiersprachen wie Ruby oder JAVA. Er kennt die Best Practices seines Bereiches und liebt es, Codes zu produzieren. Ein Entwickler muss sich von dem Gedanken lösen, ein Benutzer des selbst erstellten Systems zu sein. Er muss seinerseits lernen, mit Gestaltungsprinzipien und Mustern umgehen zu können, und sich dabei immer wieder die Fragen stellen, für wen das Produkt sein soll und welches Problem er dabei gerade löst.

Entwickler interagieren ebenfalls sehr eng mit dem Product Owner und unterstützen ihn dabei, Entscheidungen im Product Backlog zu fällen, indem sie ihn mit dem nötigen technischen Hintergrundwissen beliefern. Die Entwickler haben die Aufgabe, den Designexperten als vollwertiges Teammitglied zu akzeptieren und ein Verständnis für die Arbeitsweise des Designexperten aufzubauen. Sie lenken das Design, indem sie dem Product Owner bei der Priorisierung der Features zur Seite stehen und ihm dabei helfen, eine möglichst optimale Reihenfolge der ausgewählten Features zu erreichen.

Für die erfolgreiche Zusammenarbeit ist zu beachten, dass sowohl Designexperten als auch Entwickler unterschiedliche Lösungsansätze besitzen, wenn es darum geht, Probleme zu lösen. Dabei konzentrieren sich Designexperten auf das Wahrscheinliche, während die Entwickler eher das Mögliche fokussieren.
Darüber hinaus gibt es allerdings folgende Gemeinsamkeiten, die für eine erfolgreiche Integration dieser beiden Welten sprechen (Chaffee u. Chicoine, 2007)(Patton u. Halley, 2009):

- Erfolgreiche Produkte erstellen
- Fokus auf das bereitstellen was Menschen wertschätzen
- Iteratives arbeiten am Produkt
- User-Stories bzw. Szenarios anstatt Funktionalität

4.2. Konkrete Problemfelder und kritische Aspekte

In diesem Abschnitt werden die Problemfelder und kritischen Aspekte, die während eines Sprints eintreten können, genauer betrachtet. Hierfür werden die Unterschiede und Gemeinsamkeiten zwischen der Softwareentwicklung und dem agilen Design dargestellt.

4.2.1. Gemeinsamkeiten zwischen agilen und User-centered Designmethoden

Alistair Cockburn zieht einen Vergleich der beiden Methoden, indem er es ein kooperatives Spiel nennt: „Softwareentwicklung ist ein kooperatives Spiel von Erfindung und Kommunikation. Das primäre Ziel diese Spiels ist es, nützliche und funktionierende Software zu liefern. Das zweite Ziel ist es, sich den Rest des Spiels auf das neue Spiel vorzubereiten“ (Cockburn, 2007, S. 21ff). Es geht also darum, die Bedürfnisse der Benutzer zu verstehen, um ihnen nützliche Software zu liefern, die sie benutzen können. Designexperten versuchen ebenfalls, die Benutzer in den Vordergrund zu stellen, und suchen nach Antworten zu den Fragen über die Ziele ihrer Anwender. Also erlaubt uns UCD, ein besseres Verständnis über die Personen, die das Produkt benutzen, aufzubauen, während agile Softwareentwicklung es uns erlaubt, dieses Produkt schneller zu bauen, zu testen und auszuliefern.

Weitere Gemeinsamkeiten (Giudice, 2008):

- Einbeziehen des kompletten Teams in der Anfangsphase
- Es wird auf Benutzerfeedback während der gesamten Laufzeit Wert gesetzt
- Priorisierung hilft die Erwartungen zu erfüllen
- Flexibles Design ermöglicht Reaktion auf Veränderungen
- Kommunikation ist kritisch und muss somit transparent sein

4.2.2. Unterschiede der beiden Methoden

Der wohl größte Unterschied liegt darin, dass UCD-Methoden das Ganze betrachten, während die agile Softwareentwicklung sich auf einzelne Module und Teile des Ganzen konzentriert. Während die UCD-Methoden forschungsbasiert funktionieren, ist bei der agilen Entwicklung in erster Linie die Intuition der Entwickler relevant. Ein weiterer Unterschied liegt darin, dass beim Design Alternativen erstellt und keine Iterationen auf dem bestehenden Produkt durchgeführt werden. Im UCD-Kontext werden Informationen zu den Nutzern über User

Research eingeholt, während bei agilen Methoden die Benutzer durch konkrete Alphatests und Betatests einbezogen werden. Durch die agile Entwicklung werden Änderungen am Produkt kontinuierlich durchgeführt. Dies unterscheidet sich ebenfalls stark von der Anwendung im UCD-Umfeld; hier werden Änderungen meist planmäßig durchgeführt. Der wohl größte Unterschied und auch gleichzeitig einer der wichtigsten ist, dass Designexperten Artefakte des Produktes ausliefern, wohingegen die agilen Methoden als einem der wichtigsten Faktoren darauf aufbauen, nach jeder Iteration ein funktionsfähiges Produkt auszuliefern (Cecil, 2006).

4.2.3. Konkrete Probleme und deren Symptome

Die folgende Tabelle zeigt eine Zusammenfassung der Hauptprobleme, ihrer Symptome und der möglichen Lösungen. Sie wurde von Sy und Miller während eines „Agile UCD“ Workshops aufgestellt. Dieser Workshop hatte zum Ziel, die optimale Vorgehensweise beim „Agile UCD“ zu identifizieren (Miller u. Sy, 2007).

Probleme	Symptome	mögliche Lösungen
Nicht genug Zeit für das Design.	Entwickler warten auf das Design, Design Qualität verringert sich, Design wird nicht mit Kunden besprochen	Aufteilen der Design- und Entwicklungsarbeit auf zwei parallele Tracks, Designaktivitäten sollten möglichst klein und inkrementell sein, RITE Usability Tests, Schnelles kontextbezogenes Design, Erstellen von Design-Chunks, Verschiedene UX Aktivität in einer Sitzung miteinander kombinieren, Den Benutzer einbeziehen
Ein zu kurzer Sprint.	Design kann nicht rechtzeitig fertiggestellt werden, Keine Zeit zum überprüfen der Benutzerfreundlichkeit, Keine Zeit den Kunden zu benachrichtigen	
Ungenügend Feedback von den Benutzern.	Feedback nicht früh genug, Keine Daten mit denen man arbeiten kann — Meinungen bestimmen, Produkt ist nicht validiert	

Tabelle 4.1.: Konkrete Probleme deren Symptome und mögliche Lösungen Teil 1 von 3

Probleme	Symptome	mögliche Lösungen
Schlechter oder Schwacher Kunde.	Benutzer und Klienten wollen nicht teilnehmen, Keine Unterstützung vom Rest des Teams, Keine Sachkundigen Entscheidungen	UX Person kann als Agiler Kunde agieren, Jeder Designer arbeitet mit einem Scrum Team, Die Scrum Teams mit bedacht auswählen,
UX ist keine Vollzeit-Tätigkeit in einem agilen Team.	UX Zeit wird in viele Meetings investiert aber fliest nicht in Gestaltung oder Iterationen, Team ist durch verschlechterte UX Qualität demoralisiert	Validierte Designs werden zu den Entwicklern weitergereicht, Designer nehmen an der Sprint Plannung teil und bringen das Benutzer Feedback ein,
Keine Sprint Planung.	Großes Backlog mit viele Features und Bugs, Feedback beim Priorisieren wird ignoriert, Kontrollverlust über das timing der Designs	Keine neuen Features solange nicht andere ausgeliefert wurden
Benutzer Feedback wird ignoriert.	Feature-Set ist in Stein gegossen, Keine Zeit um Änderungen zu berücksichtigen, Neuordnung der Features ist nicht erlaubt	
Sicht auf das große Ganze ist nicht vorhanden.	Keine gemeinsame Vision oder Ziel, Zu viel Fokus auf Details, Zu schwer zum priorisieren oder Gestaltungsentscheidungen zu treffen	Agiles Team dazu überzeugen ein Sprint-Zero einzuführen, Einfache Anforderungen während des Prozesses, Zeit auf 1 oder 2 Sprint Längen verkürzen

Tabelle 4.2.: Konkrete Probleme deren Symptome und mögliche Lösungen Teil 2 von 3

Probleme	Symptome	mögliche Lösungen
Mangelhafte Kommunikation.	Missverstandene Designs, Team stimmt den Designs nicht zu, Wichtige Informationen gehen verloren	Entwickler in den Gestaltungsprozess integrieren, Benutzbarkeit in die Akzeptanzkriterien einbeziehen, Täglicher Kontakt um Fortschritt zu überprüfen, Designexperte nimmt am Daily Scrum teil, Dokumente sind für das Design Team gedacht
Nicht alle Teammitglieder sitzen nebeneinander.	Kein Teamgeist vorhanden, Kein Vertrauen, Sprach oder Zeit Barrieren, Nicht genug Kommunikation	Kommunikationswerkzeuge wie Telefon oder andere Webbasierte Werkzeuge, Gemeinsames Treffen pro Sprint
Probleme in den Abhängigkeiten.	Informationen von nicht agilen Teams wird benötigt, Dinge können nicht rechtzeitig fertiggestellt werden	Ein Scrum Moderator mit großer Überzeugungskraft kann dinge vorrantreiben

Tabelle 4.3.: Konkrete Probleme deren Symptome und mögliche Lösungen Teil 3 von 3

In den folgenden Abschnitten werden auf der Grundlage dieser Erkenntnisse die Kernpunkte untersucht, die Auslöser für einige der oben genannten Probleme sind.

4.2.4. Tätigkeitsspezifischer Sprint

Der Hauptgrund, warum der Prozess oft ins Stocken gerät und man keinen unterbrechungsfreien agilen Prozessfluss aufrechterhalten kann, verbirgt sich hinter dem Begriff „Big Design Up Front“[1][2]. Dabei werden in erster Linie detaillierte Designs erstellt, bei denen dann versucht wird, möglichst viel des „fertigen Produktes“ vorab zu gestalten und somit vorherzusehen. Diese Mentalität hat ihre Wurzeln in der Autoindustrie, wo durch ein ganzheitliches Design in der Anfangsphase hohe Änderungs- und Anpassungskosten im späteren Verlauf des Prozesses eingespart wurden.

[1] „Big Design Up Front“, url: http://en.wikipedia.org/wiki/Big_Design_Up_Front, gesehen am: 18:30, 29.11.2011

[2] Blog Artikel über den Wert von „Big Design Up Front“, url: http://agilesoftwaredevelopment.com/blog/mendelt/value-big-design-front, gesehen am: 17:00, 29.11.2011

Der Begriff beschreibt zwar eine Phase, bevor die eigentliche Entwicklung beginnt, sie hat jedoch unmittelbaren Einfluss auf den weiteren Verlauf der Sprints. Die entsprechenden Auswirkungen liegen auf der Hand: Während die Designexperten tief in der Arbeit eines vollständigen Designs stecken und dabei ungern unterbrochen werden, beginnen die Entwickler mit der Implementierung der gewünschten Features. Dies erfolgt parallel und meist ohne mit dem jeweils anderen Informationen auszutauschen. Somit ergeben sich tätigkeitsspezifische Sprints, bei denen eine Menge detailliertes Design und gleichzeitig Funktionalität erstellt werden.
Im Verlauf des Prozesses kommt es nun zwangsläufig zur Einarbeitung des bis dahin erstellten Designs mit der vorhandenen Anwendung. Dieses Design, welches hauptsächlich aus grafischen und somit statischen Dateien besteht — im besten Fall sind vielleicht HTML-Mockups vorhanden —, muss nun über die hohe Mauer zum Entwicklungsteam befördert werden. Bis hierhin wurde parallel an beiden Teilen gearbeitet, und im schlimmsten Fall erfolgte keine Synchronisation untereinander. Somit ergibt sich hier eine Problematik aus zwei verschiedenen Perspektiven. Es wurde ein umfassendes und detailliertes Design auf der Grundlage von Vorhersagen erstellt, parallel dazu wurden Features implementiert, bevor sie überhaupt gestaltet worden sind. Diese beiden Teile, die sehr orthogonal zueinander entwickelt wurden, müssen nun von den Entwicklern zusammengeführt werden. Die Entwickler besitzen jedoch nicht das nötige Hintergrundwissen zum Design, weil keinerlei Informationsaustausch oder Feedback in den einzelnen tätigkeitsspezifischen Sprints erfolgt ist.
Bevor nun mit der Arbeit begonnen werden kann, müssen sich erst einmal die beiden teilnehmenden Parteien miteinander zusammensetzen — oder besser gesagt auseinandersetzen — und das nötige Feedback einholen. Niemand kann das Resultat aus diesem Abgleich vorhersagen, jedoch kann es erhebliche Änderungen an den jeweiligen Stellen nach sich ziehen, und im schlimmsten Fall müssen ganze Komponenten erneuert werden. Ein erhebliches Problem bei tätigkeitsspezifischen Sprints besteht also darin, dass sie zu sogenannten Ende/Anfang-Beziehungen führen, bei denen eine Aufgabe abgeschlossen werden muss, bevor mit der nächsten begonnen werden kann. Meist werden dann genau bei dieser Übergabe bis dahin versteckte Aufwände sichtbar, meist ist es aber hier schon zu spät (Cohn, 2010, S. 301).

Diese Nachteile tätigkeitsspezifischer Sprints führen zu erheblichen Störungen im Prozess, weshalb man solche tätigkeitsspezifische Sprints vermeiden sollte.

- Ende/Ende statt Ende/Anfang-Beziehungen
- Implementierung/Design von zu viel oder zu wenig Funktionalität
- Implementierte Funktion/Erstelltes Design wird nicht mehr benötigt
- Implementierung/Design lässt sich sehr schwer Schätzen
- Hohe Wahrscheinlichkeit zur Terminüberschreitung
- Hoher Aufwand bei der Synchronisation beider Elemente
- Gefahr der Rückkehr zur Wasserfall Denkweise
- Kaum Priorisierungsmöglichkeiten
- Kaum Feedback
- Wenig Kommunikation
- Nicht vorhandene Transparenz

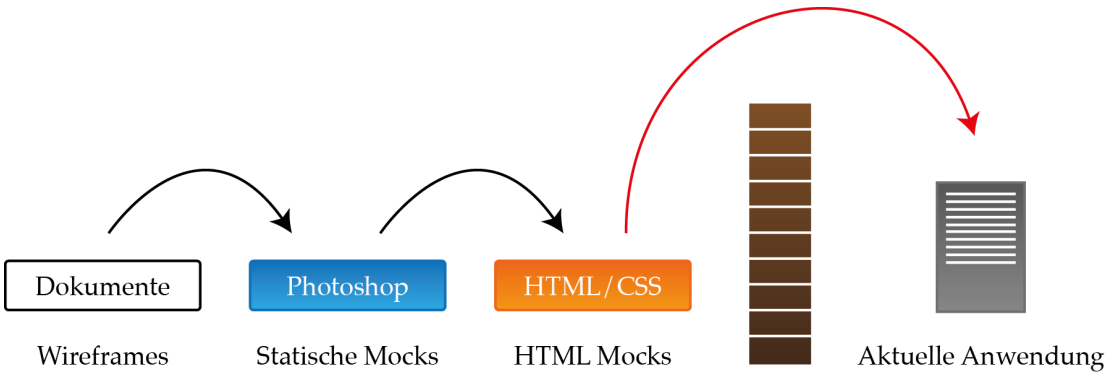


Abbildung 4.1.: Gängiger Erstellungsprozess für ein Design

Abbildung 4.1 zeigt den gängigen Erstellungsprozess eines Designs. Nach vorangegangenen, sehr detailreichen Wireframes werden statische und grafische Photoshop-Dateien erstellt und gegebenenfalls in umfangreiche HTML-Mockups umgewandelt. Danach stellt sich die Frage, wie diese Arbeit in den Code der aktuellen Anwendung über die „große Mauer“ befördert werden kann.[3]

[3] Interessante Informationen zu Feature- und Komponententeams von Mike Cohn, (Cohn, 2010, S. 213ff)

4.2.5. Kommunikationsdefizite während eines Sprints

In der heutigen Zeit sind Designexperten oftmals freiberuflich tätig; sie werden gewissermaßen zu einem neuen Projekt dazugekauft. Dieser Faktor allein macht die Integration eines Designexperten in ein Scrum-Team schon von Natur aus schwierig. Freiberuflich tätige Designexperten betreuen mehrere Projekte gleichzeitig und sind es somit gewohnt, andere Kommunikationswerkzeuge zu benutzen und aus zeitlichen Gründen auf die direkte Kommunikation zu verzichten. Der Designexperte kommuniziert somit oft nur über Tools wie E-Mail oder Telefon mit dem Kunden oder dem Entwicklungsteam. Er steht nicht während des gesamten Sprints zur Verfügung, was automatisch aus einem Prozess mit wenig Feedbackschleifen resultiert. Außerdem gibt es immer eine Verzögerung, sobald Feedback vom Entwicklungsteam eingeholt wird. Darüber hinaus durchläuft jede Gruppe auf dem Weg, ein Team zu werden, unterschiedliche Phasen. Bruce W. Tuckman hat diese Phasen in einem Modell festgehalten und spezifiziert. Dabei geht es darum, dass all diese Phasen nötig sind, um ein Team zu bilden, welches zu wachsen imstande ist, Herausforderungen annimmt, um gemeinsame Lösungen für Probleme zu finden, und folglich als hochperformantes Team Ergebnisse liefert.

Das Modell der 'Teamphasen' nach Tuckman basiert auf den folgenden 4 Phasen (Stangl, 2011):

- Forming (Formierungsphase)
  Durch Kennenlernen und gegenseitiges Einschätzen soll das Sicherheitsbedürfnis der Gruppe gestärkt werden.
- Storming (Konfliktphase)
  Die Mitglieder setzen sich auseinander und vertreten ihre Meinung, dies soll die Unterschiede der persönlichen Ziele verdeutlichen.
- Norming (Normierungsphase)
  Die Mitglieder sollen in dieser Etappe sich auf überlebensfähige Ziele Festlegen und Einigen und Abschied von Illusionen nehmen.
- Performing (Arbeitsphase)
  In dieser Phase findet die Zusammenarbeit am gemeinsamen Ziel statt, jeder muss sich engagieren und einbringen.

Diese Performanz eines Teams erreicht man nur durch Stabilität im Umfeld und im Team selbst. Nun ist diese geforderte Stabilität jedoch in dem in dieser Arbeit behandelten Fall

erheblich gestört. Durch ein externes Teammitglied müssen ständig neue Konditionen bewältigt werden, und genau aus diesem Grund leidet die Kommunikation innerhalb des Teams und im Projekt (Stangl, 2011). Sobald die direkte Kommunikation aufgrund dieser Faktoren verringert wird und sich nur schwer organisieren lässt, steigen die Kosten für die Kommunikation. Folglich resultiert daraus eine schlechtere Kommunikationsqualität, womit sich auch das Entwickeln des Produktes als sehr schwierig gestaltet. Ressourcen werden für viele neue Elemente verschwendet, die später nicht mehr verwendet werden können, weil sich elementare Anforderungen, die vorher nicht besprochen wurden, während der Projektlaufzeit verändert haben.

Alistair Cockburn beschreibt in seinem Kapitel zu „Jumping Communication Gaps“ (Cockburn, 2007, S. 94f) die Auswirkungen von Kommunikationswegen wie Telefon und E-Mail. Benutzt man das Telefon, so verzichtet man auf die visuelle Kommunikation; damit verschwinden die Körpersprache und Gestik sowie die Gesichtsausdrücke. Sobald man nur noch über E-Mail kommuniziert, so Cockburn, verzichtet man auf den Tonfall und die Möglichkeit, durch die Sprechgeschwindigkeit und Tonlage gewisse Standpunkte deutlich zu machen. Daraus resultiert offensichtlich, dass die direkte „Face-to-Face-Kommunikation der günstigste und schnellste Kanal ist, um Informationen effektiv auszutauschen.“ (Zitat: Cockburn aus (Cockburn, 2007, S. 149f))

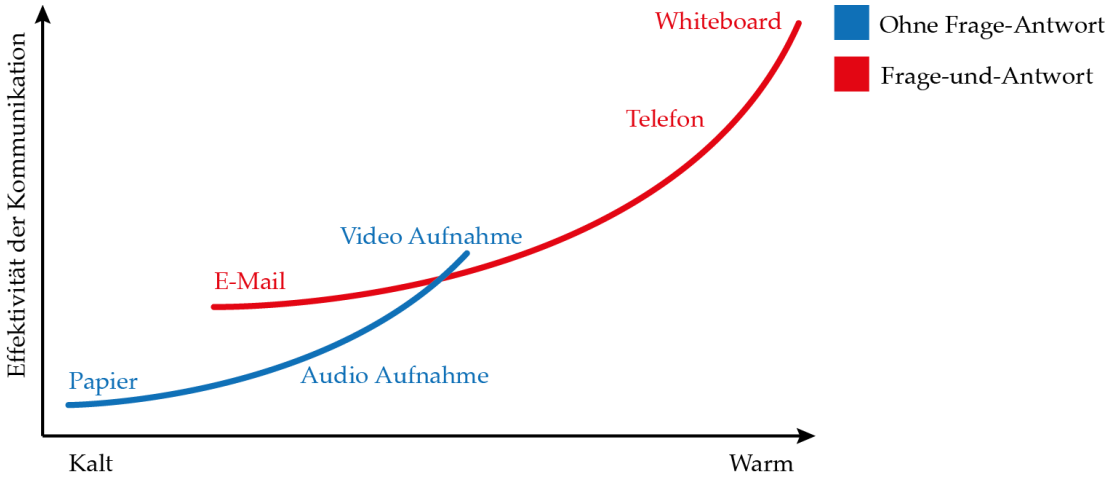


Abbildung 4.2.: Alistair Cockburn macht die unterschiedlichen Kommunikationsarten und deren Auswirkungen auf einer Temperaturskala von „Kalt“ bis „Warm“ deutlich. (Cockburn, 2007, S. 149f)

4.2.6. Behinderungen im Sprint

Aus den vorangegangenen Abschnitten leiten sich bereits diverse Problematiken ab, die negative Auswirkungen auf einen unterbrechungsfreien Prozessfluss haben. Die daraus resultierende Problematik besteht darin, dass den meisten Designexperten die entsprechenden technischen Kenntnisse über geläufige Webstandards oder Cross-Browser-Kompatibilität fehlen. Sie wissen nicht, wie Elemente in diversen Web-Browsern funktionieren oder wie aufwendig die spätere Umsetzung bestimmter Elemente ist. Entwürfe oder Prototypen werden folglich ohne dieses wichtige Hintergrundwissen erstellt und sind somit nicht selten mit großem Implementierungsaufwand und hohen Kosten verbunden. Im schlimmsten Fall müssen die vorgeleisteten gestalterischen Umsetzungen neu entworfen werden, weil festgestellt wurde, dass der Aufwand für die Umsetzung unverhältnismäßig groß ist. Tätigkeitsspezifische Sprints und die damit verbundene erschwerte Integration des Designexperten in das Scrum-Team sind Nährböden für die oben erwähnte Arbeitsweise. Aufgrund der Isolation des Designexperten entstehen gestalterische Umsetzungen, die Elemente beinhalten, für die es keine User Story gibt und die nicht mit dem Product Owner besprochen wurden. Sie müssen bei der Implementierung zwar nicht berücksichtigt werden, allerdings haben die Designexperten Zeit und Kosten in diese Elemente investiert, die an einer anderen Stelle wesentlich effektiver gewesen wären. Entsprechend wird meist von Beginn an bis zum Abschluss eines Projektes kein Wertschöpfungsprozess aufrechterhalten. Es werden überflüssige Ressourcen verschwendet, die weder einen potentiellen Wert für den Kunden liefern noch etwas zum Geschäftswert beitragen. Mary Poppendieck beschreibt in ihrem „Value Stream Mapping“[4], dass Aktivitäten in einem Prozess stets einen Wert ausliefern sollten. In ihrem Ansatz beschreibt sie, wie die Stellen im Prozess, die keinen Wert liefern und somit als überflüssig gelten, identifiziert werden können, um sie besser kontrollieren und dadurch einen effektiveren Prozessverlauf gewährleisten zu können (Poppendieck, 2011). Die Folgen, die aus dem oben skizzierten Vorgehen resultieren, sind Behinderungen und Unterbrechungen im Prozessfluss. Schlimmer noch, es werden Dinge umgesetzt, die später nicht eingesetzt werden. Oft müssen Features, die schon komplett gestaltet wurden, neu konzipiert werden, weil in der Zeit zwischen dem Entwurf des Designexperten und der Implementierung des Entwicklers Inkonsistenzen entstanden sind, die vermeidbare Kosten und Zeitinvestitionen nach sich ziehen und folglich auch den Prozessfluss behindern.

[4] Grafik zum Value Stream Mapping von Mary Poppendieck auf Mike Griffiths Homepage, url: http://leadinganswers.typepad.com/leading_answers/2011/09/pmi-acp-value-stream-mapping.html, gesehen am: 17.12.2011 um 14:30 Uhr

4.3. Mögliche Lösungsvorschläge

In den folgenden Abschnitten gilt es nun zu überprüfen, wie man die Zusammenarbeit und Kommunikation, sowohl auf technischer als auch auf menschlicher Ebene, optimieren und damit einen möglichst reibungslosen Projektverlauf erzeugen kann. Es wird darauf eingegangen, wie man durch das Zusammenspiel von Regeln und Tools ein gemeinsames Verständnis für das Produkt schafft und eine gleiche Sicht herstellt. Hierfür wird zunächst der sogenannte „Sprint-Zero“ eingeführt. Dieser ist dafür verantwortlich, dass bestimmte Vorarbeiten vor der eigentlichen Entwicklung gemeinsam erarbeitet werden, um das gemeinsame Verständnis zu etablieren. Im nächsten Abschnitt geht es dann um die kontinuierliche Zusammenarbeit während des gesamten Sprints. Es werden neue Artefakte und Etappen in den Prozess integriert, die dafür sorgen sollen, dass das Wissen im ganzen Scrum-Team möglichst optimal verteilt ist. Der letzte Abschnitt beschäftigt sich mit der Steigerung der Zusammenarbeit zwischen Designexperten und Entwicklern. Die in diesem Abschnitt vorgestellten Tools erleichtern den Informationsaustausch sowohl auf technischer als auch auf der gestalterischen Ebene und stellen somit effektive Werkzeuge dar, um die Designflexibilität zu gewährleisten und gleichzeitig eine Sicht auf das Produkt teilen zu können. Alle hier erwähnten Lösungsansätze zielen auf die zwei wichtigsten Faktoren:

Frühes Einbinden des Entwicklers in den Designprozess: Dadurch erreicht man eine höhere Identifikation mit den Benutzern, Kunden und dem Produkt an sich. Der Wissenstransfer wird gesenkt, und man erreicht eine höhere Akzeptanz der Entwickler, weil sie ihren wichtigen Input liefern und aktiv am Designprozess teilnehmen können (Beltrame, 2011).

Designexperten am Entwicklungsprozess des Produktes teilhaben lassen: Auf diese Weise erreicht man eine höhere Qualitätssicherheit für die Benutzbarkeit und das Design des Produktes. Die Entwickler haben dadurch die Möglichkeit, während des Projektes auf einen Design-Ansprechpartner zuzugreifen. Aufgrund dessen ergeben sich ein geringerer Bedarf an Refactoring und man senkt die Anzahl und den Umfang an falschen Oberflächenimplementierungen (Beltrame, 2011).

4.3.1. Kompetenzen des Designexperten

Es wurde bereits in den vorangegangenen Abschnitten erwähnt, dass Designexperten sich aus verschiedenen Gründen nur schwer in einen agilen Prozess integrieren lassen. Die in dieser Arbeit vorgestellten Lösungsvorschläge lassen sich nur dann optimal ausnutzen, wenn der Designexperte gewisse Kompetenzen mitbringt oder gewillt ist, sich diese anzueignen. Dabei spielen die technischen Kenntnisse eine relativ wichtige Rolle. Der Designexperte braucht das nötige technische Hintergrundwissen über aktuelle Web-Standards und er muss auch die gängigen Web-Browser und ihre Eigenheiten und Probleme kennen. Denn nur mit diesem Wissen im Hinterkopf kann er effektiv sein Design erstellen. Des Weiteren muss der Designexperte lernen, sein Design in kleinen Design-Chunks zu entwickeln, denn nur so lässt sich das Design in den inkrementellen Prozess einbauen. Außerdem hat der Designexperte hier den großen Vorteil, dass sein Fokus stets auf kleinen Teilbereichen liegt und er somit auch qualitativ hochwertigere Designs erstellt. Zwar ließe sich das fehlende technische Wissen durch einen Entwickler kompensieren, jedoch würde dies wiederum parallel verlaufende tätigkeitsspezifische Sprints fördern, die man unbedingt vermeiden sollte. Ebenfalls wird durch dieses technische Wissen die Hürde verkleinert, einen funktionsfähigen Frontend-Code anstelle von statischen Photoshop-Dateien zu erstellen. Bei statischen Dateien tritt die größte Arbeitslast gegen Ende auf, wenn es um die Einarbeitung in die Anwendung geht, ein funktionsfähiger Frontend-Code hingegen hält die Belastung kontinuierlich relativ gering, weil die Entwickler dadurch validierte und funktionsfähige Designs mit geringem Implementierungsaufwand erhalten. Betrachtet man sowohl die Benutzerziele als auch die Geschäftsziele, hilft dies, das Verständnis zu verbessern, wie man ein Produkt entwickeln sollte, damit beide Ziele erfüllt werden können. Designexperten sind meist gut darin zu wissen, was die Benutzer möchten, brauchen oder akzeptieren, aber weniger gut darin, was die Geschäftsziele sind. Aufgrund dessen ist eine enge und kontinuierliche Zusammenarbeit mit dem Product Owner unvermeidbar.

Verantwortlichkeiten eines Designexperten im agilen Prozess:

- Benutzerfeedback einsammeln und für kommende Implementierung vorbereiten
- Design für kommende Iteration erstellen
- Als Designcoach für die Entwickler im aktuellen Sprint verfügbar sein
- Testen der implementierten Features im vorherigen Sprint

Der Usability-Experte Jakob Nielsen sagt: „Die größte Umstellung für Oberflächenexperten in agilen Teams betrifft die Denkweise. Gute allgemeine Kenntnisse über User Experience zu haben, ist hilfreich, um die herkömmlichen Design- und Bewertungsmethoden zu ändern, so dass sie mit der Schwerpunktsetzung in agilen Teams übereinstimmen. Letzten Endes jedoch müssen sie sowohl an sich selbst glauben als auch die Prinzipien der agilen Entwicklung verinnerlichen, wenn sie Erfolg haben wollen.“ (Zitat: Jakob Nielsen aus (Cohn, 2010, S. 181)). Wenn Designexperten also bereit sind, die Vorgehensweisen zu verändern und Verantwortung zu übernehmen, bieten sich ihnen hervorragende Möglichkeiten, um sich effizient und nachhaltig in einem agilen Team zu platzieren und an Bedeutung zu gewinnen.

4.3.2. Einführung des ‘Sprint-Zero’

Scrum sieht vor, dass in jedem Sprint potentiell lieferbare Software produziert wird, die einen Geschäftswert mit sich bringt. Der Sprint-Zero ist hier die Ausnahme. Man benutzt diesen Sprint dafür, ein Team zu formen, bevor man beginnt, ein Produkt zu bauen. Es ist offensichtlich, dass man kein fertiges Team innerhalb einer Woche aufbauen kann, man kann aber damit anfangen. Denn genau dieser Anfang kann später den entscheidenden Unterschied ausmachen, der darin besteht, dass ein geschlossenes Team das Produkt entwickelt anstelle einer uneinheitlichen Gruppe von Einzelpersonen.
Zu Beginn eines solchen Sprints geht es in erste Linie darum, dem Team und vor allem dem Kunden die agilen Methoden und Scrum näherzubringen. Gleichzeitig wird hierbei darauf Wert gelegt, den Teamgeist zu stärken, so dass man seinen Teampartner besser kennenlernt. Im weiteren Verlauf dieses Sprints wird die Produktvision (siehe Abschnitt 5.1.1) in einem sogenannten Elevator-Pitch (siehe Abschnitt 5.1.3) mit allen Beteiligten erstellt. Hier wird der Grundstein für das gemeinsame Verständnis und die Ziele des Produktes gelegt und die Anforderungen werden festgehalten. Darauf aufbauend werden nun vom Product Owner zusammen mit den restlichen Teammitgliedern die einzelnen Features in User Stories überführt und in das Product Backlog geschrieben. Dabei erhält der Kunde bereits in dieser Phase Feedback und Unterstützung der Entwickler im Hinblick auf technische Machbarkeit und Umfang. Der Product Owner sollte hierbei in der Lage sein, möglichst ausreichend Backlog-Einträge für den ersten oder die ersten beiden Sprints zu generieren.
Sind nun genügend Einträge im Product Backlog vorhanden und priorisiert, haben die Entwickler zusammen mit dem Designexperten die Aufgabe, diese Einträge zu schätzen und eventuell in mehrere kleinere Stories aufzuteilen. Der nächste Schritt besteht darin, eine „Definition of Done“ zu erstellen. Hierbei einigen sich Entwickler und Product Owner, wann

eine User Story abgeschlossen ist und abgenommen werden kann. Des Weiteren wird die Länge eines Sprints unter allen Projektteilnehmern geklärt und festgesetzt.
Der letzte Teil des Sprint-Zero zielt darauf ab, alle vorangegangenen Aktivitäten komplett abzuschließen, damit effektiv und ohne weitere Unterbrechung mit dem ersten Sprint und somit mit der Implementierung des Produktes begonnen werden kann. Ebenfalls muss hier noch einmal auf die Team-Building-Maßnahmen eingegangen werden, um den Teamgeist zu vertiefen, wozu es viele interessante Übungen und Spiele gibt.

4.3.3. Zusammenarbeit während des gesamten Sprints

Bis hierhin wurden alle Hintergründe, die für eine erfolglose Integration der Designexperten in den agilen Prozess verantwortlich sind, aufgezeigt. Des Weiteren wurde schon auf konkrete Lösungsmechanismen eingegangen, die nun in den folgenden Kapiteln vertieft werden. Die agilen Methoden bieten bereits gute Möglichkeiten einer Zusammenarbeit von Entwicklern und Designexperten. Jedoch müssen zur Unterstützung neue Artefakte und Etappen in den Prozess integriert werden, damit der Designexperte enger am Team gehalten wird und über die gesamte Projektlaufzeit hinweg mit den Entwicklern zusammenarbeiten kann. Durch die regelmäßige Beteiligung an den hier vorgeschlagenen Etappen und unter Verwendung der hier aufgezeigten Artefakte wird nicht nur die gemeinsame Kommunikation gesteigert, sondern die räumliche Trennung verringert. Dies führt zu einem wesentlich schnelleren Feedback als bisher. Die Kommunikation zwischen den beiden Parteien ist stets kritisch, weshalb sie eine höhere Transparenz aufweisen muss. Beide müssen schon früh anfangen miteinander zu kommunizieren und möglichst oft zusammenarbeiten. Dieses Vorgehen erfordert ein hohes Maß an Disziplin.

Regelmäßige Teilnahme des Designexperten am Daily-Scrum- und Konzeptionsmeeting
Daily Scrums werden nur selten von Designexperten zu deren Vorteil genutzt. Der Designexperte erhält hier die Möglichkeit, sich mit den Entwicklern zu synchronisieren und evtl. Behinderungen zu erwähnen. Somit ist er immer auf dem laufenden Entwicklungsstand und kann bereits hier das von den Entwicklern dringend benötigte Feedback geben. Auf der anderen Seite kann er ein gründliches Verständnis der technischen Einschränkungen gewinnen, die die Gestaltungsentscheidungen betreffen.
Das Konzeptionsmeeting bringt für beide Seiten Vorteile mit sich. Auf der Entwicklungsseite wird wöchentlich dafür gesorgt, dass die für den kommenden Sprint relevanten Einträge im Product Backlog mit Details versehen sind, die zu Beginn des nächsten Sprints für eine effektive Schätzung nötig sind. Auf der anderen Seite erhält der Designexperte einen

Überblick darüber, welche Design-Chunks für den nächsten Sprint von höchster Relevanz sind. Somit erhalten beide Parteien durch die im Konzeptionsmeeting erstellten Prototypen und Scribbles eine bessere Sicht auf das große Ganze (Ramsay, 2011a).
Die Hauptakteure eines Konzeptionsmeetings sind der Product Owner und der Designexperte. Beide Sparringspartner erzeugen quasi im argumentativen Schlagabtausch gemeinsam grobe Wireframes und HTML-Mockups der bevorstehenden Design-Chunks. Diese müssen nur insoweit detailliert genug sein, dass es zu einer konstruktiven Kommunikation sowie einem gemeinsamen Verständnis beiträgt und der Lernerfolg gefördert wird. Dabei stehen ihnen die Entwickler für technische Fragen zu Umsetzung und Aufwand zur Seite. Es entstehen wichtige Konversationen, bei denen die Entwickler zum Beispiel eine alternative Implementierung vorschlagen können, die zwar äußerlich identisch aufgebaut werden kann, jedoch einen wesentlich geringeren Implementierungsaufwand mit sich bringt. Die in diesem Meeting erstellten Scribbles werden dann für den weiteren Projektverlauf für alle Teammitglieder und Projektbeteiligten zugänglich und ersichtlich hinterlegt und dokumentiert, inklusive aller Notizen und getroffenen Entscheidungen zu diesem Scribble.
Ein weiterer Teil des Konzeptionsmeetings wird dafür verwendet, dass Designexperten ihre konkreten Designentscheidungen aus vorangegangenen Meetings kurz präsentieren und nach Feedback und Machbarkeit in der Entwicklung fragen. Es werden ebenso Resultate aus den Benutzertests präsentiert, so dass jeder über die eventuell auftretenden Probleme mit dem Interface aufgeklärt ist und seine Vorschläge äußern kann. Das Konzeptionsmeeting hat also zum Ziel, am Ende eines jeden Meetings grobe Skizzen der bevorstehenden Funktionalität zu liefern, die unter der Leitung von Designexperte und Product Owner mit allen Projektbeteiligten erstellt werden. Am Ende dieses Meetings sind beide Parteien über die gegenseitigen Herausforderungen aufgeklärt, außerdem herrscht eine gemeinsame Sicht auf die zu entwickelnde Funktion.

Folgende Vorteile können aus diesem Konzeptionsmeeting gezogen werden:

- Designexperten arbeiten eng mit dem Product Owner zusammen
- Das Produktkonzept wird stets validiert und artikuliert
- Ganzheitliches Denken wird gefördert
- Anwendung des inkrementellen Designs wird Unterstützt
- Verständnis der Entwickler, für das hinter dem Design liegende Produkt, wird gestärkt

- Relevante Interaktionen und Design-Muster werden hier deutlich gemacht und diskutiert
- Das Verständnis für Designentscheidungen wird gestärkt
- Zeitersparnis für Entwickler durch verantwortungsbewusste Entscheidungen

Was die Entwickler können, können die Designexperten ebenfalls — Pair Designing Inspiriert durch die XP-Technik des Pair Programming kommt beim agilen Design das Pair Designing zum Einsatz. Hierfür setzt sich der Designexperte mindestens einmal in der Woche zusammen mit einem Entwickler an einen Computer, um entweder gemeinsam an einer aktuellen User Story mit relativ hohem Frontend-Anteil zu arbeiten oder liegengebliebene Frontend-Bugs oder Refactorings durchzuführen. Als Grundlage hierfür wird der interaktive HTML-Styleguide verwendet, auf den in Kapitel 5 eingegangen wird. Der Designexperte lernt hier die nötigen Techniken kennen und stärkt sein Gefühl und Verständnis für die zugrunde liegende Technologie (Ramsay, 2011b).

Routinearbeit: Kontinuierlicher Design Feedback Task Besonders für eine effektive Zusammenarbeit ist das kontinuierliche Feedback des Designexperten wertvoll. Es wird ein Ort oder ein Medium definiert, auf dem der Designexperte während des Sprints sein Feedback zur implementierten Oberfläche festhält. Der Entwickler kann somit zu jeder Zeit darauf zugreifen und die dort festgehaltenen Anregungen oder Wünsche umsetzen. Falls Unklarheiten auftreten, kann der Entwickler gegebenenfalls noch einmal Rücksprache mit dem Designexperten halten. Dieses Design Feedback kann ebenfalls Grundlage einer Pair Designing Session werden (siehe Abb. 4.3).

Erstellen von „Design-Chunks“ Design-Chunks sind kleine Designeinheiten, die sich während eines Sprints leicht realisieren lassen. Sie erweitern inkrementell das Gesamtdesign der Anwendung. Die Anwendung und Benutzung der im Kapitel 5 aufgeführten Werkzeuge führt dazu, Design-Chunks leichter und effektiver realisieren zu können. Für kleine Designeinheiten lassen sich z. B. die Prototypen wesentlich schneller erstellen als für komplexe Oberflächen. Sy sagt: „Interaktionsdesigner sind darin geschult, Oberflächen ganzheitlich zu betrachten, so dass die Zerlegung des Designs in Teile — noch dazu in Teile, die für sich selbst keinen Arbeitsablauf darstellen — zunächst schwierig ist. Die Aufteilung des Designs in Abschnitte bringt viele Vorteile.“ Einer der Vorteile liegt darin, dass keine unnötige Funktionalität umgesetzt wird, die man später nicht mehr benötigt. Ein wesentlicher Vorteil ist auch, dass man durch Designeinheiten ein „Just-in-time“-Design (Sy, 2007, S. 116) realisiert,

d. h., man hat immer genau den richtigen Umfang an Design passend zu der aktuellen Funktionalität (siehe Abb. 4.3). Daraus resultiert laut Sy eine wesentlich bessere Gestaltung (Sy, 2007, S. 120ff)(Cohn, 2010, S. 303). Zu lernen, kleine Designeinheiten während eines Sprints zu erstellen und diese auszuliefern, ist eine der ersten großen Hürden des agilen Designs. Jedoch erreicht man durch die einzelnen Designeinheiten jedes Mal eine genauere Detailstufe und arbeitet sich sozusagen Stück für Stück an das Endresultat.[5]

4.4. Zusammenfassung

Die in dieser Arbeit vorgeschlagene Optimierung des Prozesses fördert die tägliche Zusammenarbeit zwischen den Entwicklern und den Designexperten. Sie ist essentieller Bestandteil der erfolgreichen Einführung von „Agile Design“. Außerdem wird die Kommunikation zwischen den Beteiligten transparenter und trägt zur Verbesserung des Produktes bei.

4.4.1. Optimierter Prozessverlauf

Sprint-Zero

Das Projekt startet mit einem „Sprint-Zero“, welcher dazu dient, möglichst ein gemeinsames Verständnis des Produktes aufzubauen und dessen Anforderungen zu verstehen. In diesem Vorab-Sprint werden elementare Dinge geklärt und versucht, die Projektbeteiligten zu einem Team zu formen. Der Sprint-Zero beinhaltet alle Aktivitäten, die einen reibungslosen Projektstart mit dem ersten Sprint gewährleisten. Hierbei wird auch im sogenannten „Elevator-Pitch“ die Produktvision erstellt, auf deren Basis dann das Product Backlog mit entsprechenden User Stories gefüllt und priorisiert wird.

Sprint 1 - N

Der weitere Projektverlauf basiert letztendlich auf dem Grundsatz, dass es nicht der Designexperte ist, der das Design erstellt, sondern das Team. In einem Sprint haben sowohl die Designexperten als auch die Entwickler unterschiedliche Aufgaben, die sie abarbeiten müssen. Die Designexperten testen und validieren die Oberflächenimplementierungen des vorherigen Sprints. Sie unterstützen die Entwickler im aktuellen Sprint bei der Implementierung der Oberflächen, indem sie diese zusammen mit dem Entwickler beim Pair Designing unter Zuhilfenahme des interaktiven Styleguides erarbeiten. Außerdem gestalten

[5] Interessanter Blog Artikel über „Twelve emerging best practices for adding UX work to Agile development“ von Arlen Bankston (2008), url: http://www.agileproductdesign.com/blog/emerging_best_agile_ux_practice.html , gesehen am: 30.10.2011 um: 18 Uhr

sie die Oberflächen für den kommenden Sprint. Hierbei ist es wichtig, dass sie dennoch im Hinterkopf behalten, das Design aufgrund von Feedback anpassen oder ändern zu können. Die Entwickler hingegen haben zur Hauptaufgabe, die für den aktuellen Sprint vorgesehene Funktionalität zu implementieren und dabei stets auf das Feedback des Designexperten zuzugreifen. Die für den aktuellen Sprint in Frage kommenden Elemente der Anwendung werden gemeinsam in einem Konzeptionsmeeting vor jedem neu startenden Sprint besprochen und gemeinsam erarbeitet. So wird aus unklaren Anforderungen Stufe für Stufe ein konkretes Stück Software gebaut (siehe Abb. 4.3).

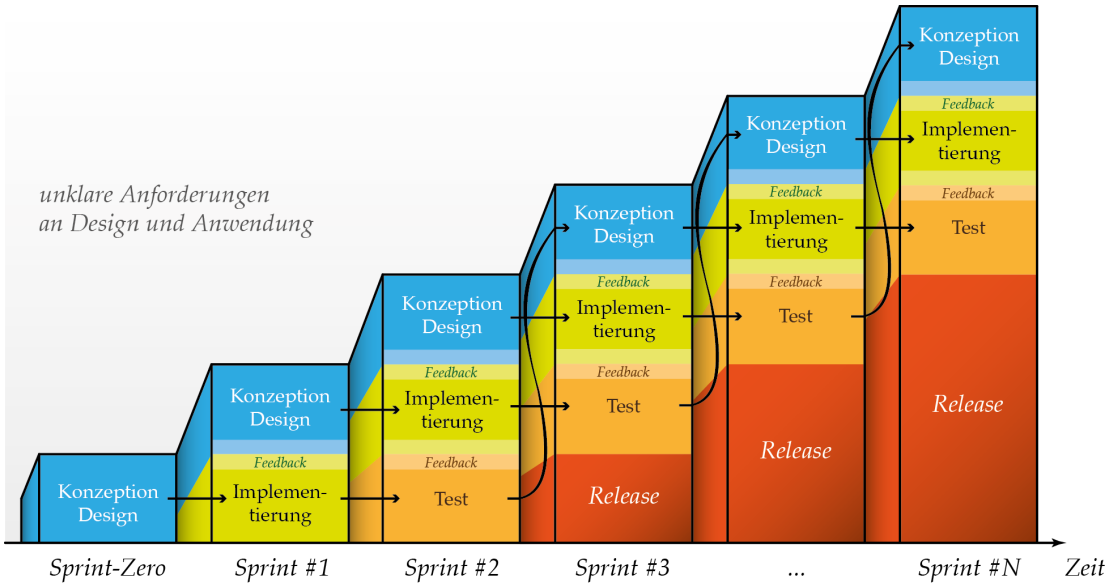


Abbildung 4.3.: Inspiriert durch das „Step-Ladder model“ (Ramsay, 2011a) zeigt die Grafik wie die parallele aber gleichzeitige Zusammenarbeit während der Projektlaufzeit aussehen sollte, um möglichst Effektiv ein inkrementelles Design durchführen zu können

Durch Anwendung der aufgeführten Aktivitäten lässt sich ein kontinuierliches und inkrementelles Design erreichen, welches parallel zu und zeitgleich mit der aktuellen Implementierung entwickelt wird und sich folglich leichter anpassen lässt. Man erzeugt immer genau so viel Design, wie man gerade für die Anwendung benötigt. Desirée Sy nennt dies in ihrer Arbeit das „Just-in-time“-Design (Sy, 2007, S. 116). Der Designexperte bekommt einen besseren Einblick in die zugrunde liegende Architektur und Technologie, somit kann er auch wesentlichen Einfluss auf die Implementierung der Oberflächen nehmen. Daher ist eine starke Zusammenarbeit zwischen allen Projektbeteiligten sehr wichtig und von hohem Nutzen, sie erfordert jedoch gleichfalls ein hohes Maß an Disziplin und Ehrlichkeit. Es ist

nichts Falsches daran, das Design vorweg zu entwickeln, man sollte nur stets darauf achten, dass die Entwickler und der Product Owner über die aktuellen Tätigkeiten informiert sind und den Fortschritt überprüfen können. Ungeachtet dessen, dass Designexperten relativ viel Zeit für vorausschauende Tätigkeiten verwenden, ist es wichtig zu beachten, dass sie stets Mitglieder des Teams sind, welches am aktuellen Sprint arbeitet (Nebe u. a., 2007).

4.4.2. Vorteile und Gefahren für Entwickler und Designexperten

Folgende Vorteile und Gefahren ergeben sich aus dem optimierten Prozessverlauf für beide Seiten, sowohl für die Designexperten als auch für die Entwickler[6]:

Vorteile für Designexperten:

1. Keine Zeitverschwendung für später nicht genutzte Designs
2. Zeitersparnis durch gleichzeitige Durchführung der Überprüfung der Features
3. Rechtzeitiges Feedback während des gesamten Projektlaufzeit
4. Schnelle Reaktionszeit auf Veränderungen durch Chunks
5. Wissen über die zugrunde liegende Technologie

Vorteile für Entwickler (Patton, 2009):

1. Maximierung der Implementierungszeit, weil nicht auf Designtätigkeiten gewartet werden muss
2. Keine Zeitverschwendung für die Implementierung unterschiedlicher Design Varianten
3. Entwickler lernen gestalterische Muster und deren Einsatz kennen

Es besteht jedoch auch die Gefahr der Rückkehr zur Wasserfall-Denkweise und Parallelverarbeitung (Cockburn, 2007):

- Die Designexperten arbeiten dem Entwicklungsteam voraus, machen sich aber selbst nicht für die Entwickler verfügbar.
- Die Designexperten scheitern bei der Auswertung und Einarbeitung des Feedbacks aus vorangegangenen Sprints.

[6] Interessanter Blog Eintrag über „Designing Ahead: The Good, The Bad, and The Ugly“ von Anders Ramsay (2010), url: http://www.andersramsay.com/2010/08/22/designing-ahead-the-good-the-bad-and-the-ugly, gesehen am: 20.11.2011 um 18 Uhr

- Das Feedback wird aus zeitlichen Gründen nach hinten verschoben und fließt zu spät ein.
- Die Diskussionen, die vorher wie Gemeinschaftsarbeiten gewirkt haben, werden nun als Unterbrechung bei der Arbeit empfunden; um sich selbst zu schützen, schreiben die Designexperten Dokumente oder hinterlegen Bilder der zu implementierenden Oberflächen für die Entwickler.

Dabei gibt es durchaus Praktiken zur Vermeidung der Parallelverarbeitung. Laut Lynn Miller (Miller, 2005) lässt diese sich vermeiden, indem man in der Vergangenheit, Gegenwart und der Zukunft arbeitet und somit die Zusammenarbeit aller Teammitglieder und das Feedback zwischen Design und Entwicklung aufrechterhält.

Vergangenheit	**Gegenwart**	**Zukunft**
Die meisten Designs mussten aufgrund von technischen Probleme oder Problemen bei der Implementierung vorsichtig verbessert werden, dabei zeigten die Usability Tests nicht wie die einzelnen Funktionalitäten mit einander interagieren	Die Designexperten arbeiten täglich mit dem Entwicklungsteam zusammen um während der gesamten Implementierungsphase fragen seitens der Entwickler beantworten zu können und Probleme lösen zu können die bei der Implementierung entstehen können	Designs werden nicht einfach über die mauer zu den Entwicklern geworfen. Durch die Daily Scrums und die Präsentationen der Oberflächen bei regelmässigen Meetings können die Entwickler dem Designfortschritt folgen

Tabelle 4.4.: Praktiken zur Vermeidung der Parallelverarbeitung (Miller, 2005)

4.4.3. Die 14 Prinzipien des Agile Designs

1. Designexperten sind vollwertige Mitglieder des Scrum Teams
2. Das gesamte Scrum Team erstellt zusammen mit dem Product Owner eine Produktvision
3. Die Entwickler werden so früh wie möglich in denn Designprozess eingebunden
4. Designexperten führen zusammen mit dem Product Owner das Konzeptionsmeeting an
5. Resultat eines Konzeptionsmeetings ist stets eine grobe Skizze der besprochenen Oberfläche
6. Das Design wird Stück für Stück inkrementell entwickelt
7. Designexperten arbeiten genau einen Sprint vorweg
8. Bei komplexen Interaktionen oder Eingabemasken werden Papierprototypen angewandt
9. Design-Feedback, Umsetzung oder Bugs werden so oft es geht im Pairing durchgeführt
10. Designgrundlage ist die aktuelle Ansicht im Browser und nicht die statische Photoshop-Datei
11. Zeit für Design kann durch komplexe Implementierung-Stories erkauft werden
12. User-Stories beinhalten implizit die Umsetzung der Oberfläche dafür, es gibt keine extra Design Story dafür
13. Der Designexperte steht den Entwicklern stets als Designcoach und Ansprechpartner zur Seite
14. Agile Design erfordert ein hohes Mass an Disziplin und Ehrlichkeit zusätzlich zu den Werten der agilen Softwareentwicklung

5. Realisierung

Um einen reibungslosen Prozessverlauf gewährleisten zu können, muss man nicht nur die Zusammenarbeit durch Steigerung der Kommunikation bereichern, sondern auch die entsprechenden Werkzeuge dafür haben, die die Grundlage für eine effektive Zusammenarbeit bieten. Einerseits muss das Bild vom großen Ganzen von den Projektbeteiligten geteilt werden, auf der anderen Seite müssen sich Entwickler und Designexperten untereinander verstehen. Hierfür dienen bestimmte Werkzeuge, die im Projektverlauf eingesetzt werden. Was man durch Einsatz dieser Werkzeuge ebenfalls erreichen will, ist die Flexibilität des Designs in der Anwendung. Man kann mit Hilfe dieser Werkzeuge auf Veränderungen, Weiterentwicklung oder Anforderungen der Anwendung besser reagieren.
Die Werkzeuge haben sich insoweit bewährt, als sie die Kommunikation und das Verständnis steigern, man darf sich jedoch nicht auf ihren Funktionen ausruhen. Sie funktionieren nur optimal, wenn man sie gemeinsam nutzt und dabei Informationen und Feedback austauscht. Durch korrekte Anwendung dieser Werkzeuge erreicht man ein konsistentes Design über die gesamte Anwendung hinweg. Des Weiteren halten sie den Implementierungsaufwand während der einzelnen Sprints relativ niedrig, so dass der Prozess nicht ins Stocken gerät, sobald es darum geht, das Design zu implementieren.

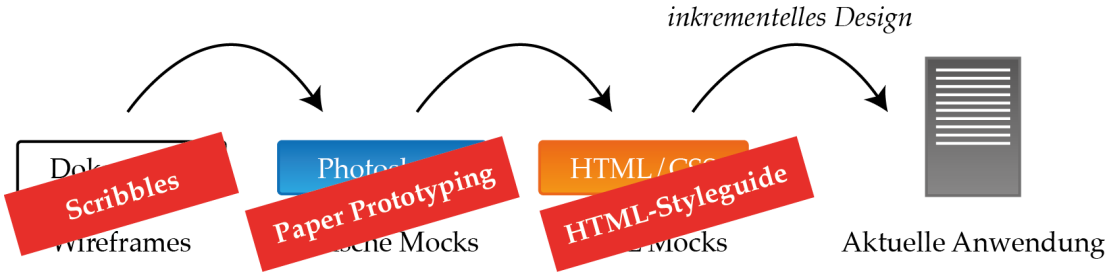


Abbildung 5.1.: Diese Grafik zeigt die positiven Auswirkungen bei korrekter Anwendung inkrementellen Designs (Abb. 4.3) im Vergleich zu gängigen agilen Prozessen (vgl. Abb. 4.1)

5.1. Produktvision

Bei einem hoch iterativen und explorativen Prozess wie bei Scrum ist es zwingend notwendig, die Aufmerksamkeit auf die Wahl der Ziele und Bedürfnisse zu richten, damit die Anfangsbedingungen allen Projektbeteiligten bekannt sind. Diese müssen so gewählt werden, dass sie weder zu starr noch zu beliebig formuliert sind, sondern gerade so, dass sie Orientierung und den nötigen Stabilitäts- und Inspirationsrahmen für alle Aspekte der Entwicklung bieten.
Werden sowohl die Benutzerziele als auch die Geschäftsziele verstanden, hilft dies allen Projektteilnehmern, ihr Verständnis zu verbessern, wie dieses Produkt entwickelt werden kann, so dass beide Ziele erfüllt werden können. Designexperten sind meist gut darin zu wissen, was die Benutzer möchten oder brauchen, aber weniger gut darin, was die Geschäftsziele sind. Unter anderem aus diesem Grund ist es wichtig, dass Designexperten zusammen mit den Entwicklern und dem Product Owner die Produktvision erstellen.

5.1.1. Was ist eine Produktvision?

Eine Produktvision ist die gröbste Zeichnungsebene des fertigen Produktes und gibt somit das Ziel des Produktes vor. Sie sollte Antworten auf die folgenden drei Fragen liefern:

1. Wohin wollen wir gehen?
2. Wie kommen wir dahin?
3. Warum werden wir erfolgreich sein?

Die Vision gibt also nicht nur das Ziel vor, sondern auch den Weg dorthin und weshalb man meint, auf diesem Weg auch wirklich zum Ziel zu kommen. Die Antwort auf die erste Frage umfasst die grundlegenden Eigenschaften des neuen Produktes, die Antwort auf die zweite Frage das, was getan werden muss, um das Produkt zu erstellen und im Markt zu platzieren. Und die Antwort auf die letzte Frage gibt an, was an dem Produkt anders ist, welche Probleme es löst, die bisher ungelöst sind, oder was es besser macht als andere bereits existierende Produkte. Der Erfolg kann aber auch durch die Erschließung eines neuen Marktes kommen. Die Produktvision ist der Startpunkt eines Projektes und sollte relativ kurz sein, damit sie von allen schnell erfasst werden kann.

Warum gute Anforderungen wichtig sind, erläutert Aza Raskin in seinem Artikel „Iterative Design: Towards the Perfect Paper Plane“. Dort beschreibt er, dass zu abstrakt formulierte Anforderungen zu viel Spielraum für die Lösung lassen, die besonders im Hinblick auf das

selbst organisierte Scrum-Team als wichtig zu betrachten ist. Bei einer Anforderung wie „Designe ein besseres Auto“ würde die ausgeführte Lösung sich nie diesem gewünschtem Ziel annähern — es gibt einfach zu viele Möglichkeiten, wie man „besser“ interpretieren kann. Eine gut formulierte Problemdarstellung bietet eine überschaubare Menge an Einschränkungen, mit denen man beginnen kann. Aufgabe des iterativen Vorgehens ist es, diese Einschränkungen Schritt für Schritt zu verfeinern. In dem genannten Artikel findet sich auch ein Zitat von Marissa Mayer, Google VP for User Experience wieder. Es beschreibt den befreienden Nebeneffekt von Einschränkungen:
„When people think about creativity, they think about artistic work — unbridled, unguided effort that leads to beautiful effect. But if you look deeper, you'll find that some of the most inspiring art forms, such as haikus, sonatas, and religious paintings, are fraught with constraints. They are beautiful because creativity triumphed over the 'rules'. Constraints shape and focus problems and provide clear challenges to overcome. Creativity thrives best when constrained“ ((Mayer, 2006) zitiert nach (Raskin, 2007)).

Die Kerneigenschaften einer guten Produktvision sind die folgenden:

- Sie wird akzeptiert und ist mit dem gesamten Team erarbeitet worden.
- Sie ist stabil und klar formuliert.
- Sie ist motivierend und lässt Freiraum für das Team.
- Sie ist kurz und bündig, damit sie von allen schnell erfasst werden kann.
- Sie ist während der gesamten Projektlaufzeit für alle sichtbar.

5.1.2. Warum braucht man eine Produktvision?

Weshalb braucht man nun eine solche Produktvision? Reicht es nicht, wenn der Product Owner oder die Geschäftsführung ein Verständnis davon haben? Scrum sieht sich selbst organisierende Teams vor. Gäbe es nun keine Produktvision, könnte sich das Team im Rahmen der Selbstorganisation irgendwohin bewegen. Mit der Produktvision befreit man das kreative Potential des Teams und gibt ihm eine Richtung vor. Die Vision hilft somit auf der einen Seite, kleine Entscheidungen im Sinne des großen Ganzen zu treffen, auf der anderen aber ermöglicht sie auch, schwerwiegendere Entscheidungen, die z. B. die Architektur betreffen, mit Blick auf das Endergebnis in einen Gesamtzusammenhang zu bringen. Folglich ergeben sich aus einer Produktvision natürliche Sortierkriterien für das Backlog. Ohne Vision

könnten bestimmte technische Backlog-Einträge für den Markt wichtiger als andere erscheinen; hier dient die Vision als Priorisierungshilfe.
Die Vision zieht sich wie ein roter Faden durch das Projekt. Sie dient der Fokussierung und der Motivation innerhalb des Teams. Nicht zuletzt kann sich der Product Owner ihrer bedienen, um das Management innerhalb der eigenen Organisation zu überzeugen. Je mitreißender und eingängiger die Vision ist, umso leichter wird es für den Product Owner. Sie ist somit nicht nur richtungsweisend für das Produkt, sondern auch das Herz eines Projekts. Um die zentralen Artefakte und Ideen von Scrum wie das Product Backlog, die Selbstorganisation, die Eigenverantwortlichkeit des Teams und die enge Zusammenarbeit mit dem Kunden realisieren und managen zu können, braucht man eine gemeinsame Idee zwischen allen Projektbeteiligten. Auch die Designexperten zählen zu diesen Beteiligten (Buchner, 2008).

5.1.3. Der Elevator Pitch

Die größte Herausforderung bei der Formulierung einer Produktvision liegt darin, sie auf der richtigen Abstraktionsebene zu platzieren. Das folgende Zitat von Carl Schurz macht dies sehr gut deutlich: „Ideale sind wie Sterne. Man kann sie nicht erreichen, aber man kann sich nach ihnen orientieren“ (Zitat: Carl Schurz). Aus diesem Grund sollte die Produktvision nicht mehr tun, als zwei einfache Fragen zu beantworten — „Was will das Unternehmen?“, „Was will der Kunde?“ — und damit einen ebenso klaren wie flexiblen Rahmen für die Entwicklung zu definieren.
Der Elevator Pitch ist der klassische Weg, die Produktvision zu validieren, indem man die Anforderung stellt, das Produkt in der kurzen Zeit einer Fahrstuhlfahrt (ca. 30–50 Sekunden) beschreiben zu können. Wird dieser Test bestanden, ist gewährleistet, dass die Produktvision klar, verbindlich und kurz genug ist. Zu beachten ist dabei, dass der Elevator Pitch keine Rückschlüsse darauf liefert, ob die richtigen Benutzerbedürfnisse und Produkteigenschaften ausgewählt worden sind. Nur ein frühes Benutzerfeedback kann dies garantieren (Pichler, 2009).

Die prägnanteste Formulierung der kundenbezogenen Aspekte der Produktvision findet sich in einem Elevator Pitch wieder, der folgende Elemente enthält:

> **For** *[target customer]*
>
> **who** *[statement of need or opportunity]*
>
> **the** *[product name]*
>
> **is a** *[product category]*
>
> **that** *[key benefit, compelling reason to buy].*
>
> **Unlike** *[primary competitive alternative],*
>
> **our product** *[statement of primary differentiation].*

Diese Formulierung wird ergänzt durch eine Darstellung der Geschäftsziele, die hinter der Entwicklung stehen. Diese werden durch eine SMART-Analyse untersucht. Geschäftsziele geben dem Unternehmen eine klare Vorgabe; daraufhin können Pläne erstellt werden, um diese Vorgabe zu erreichen. Es ermöglicht dem Unternehmen, den Fortschritt auf diesem Weg zu messen (tutor2u, 2011).

Effektive Geschäftsziele sollten den folgenden Kriterien entsprechen (tutor2u, 2011):
Specific — Geschäftsziele sollten möglichst spezifisch sein.
Measurable — Geschäftsziele liefern einen messbaren Wert für das Unternehmen.
Agreed — Geschäftsziele sollten von allen beteiligten akzeptiert werden.
Realistic — Geschäftsziele sollten erreichbare Herausforderungen darstellen.
Time specific — Geschäftsziele sollten in bestimmter Zeit erreicht werden.

5.2. Von der Produktvision zum Produkt

Wie kommen wir nun von der Produktvision, also der abstrakten Formulierung der Wünsche und Bedürfnisse eines Unternehmens und der Benutzer, zu konkreten Anforderungen und Features und somit im weiteren Verlauf zu einem konkreten Produkt? Joshua Porter schlägt dafür die sogenannte AOF-Methode vor, aus der sich später die für das Produkt relevanten User Stories ableiten lassen.
Ebenso wie die Produktvision ist das gemeinsame Erarbeiten und Erstellen von User Stories ein elementares Werkzeug für die Projektbeteiligten. Effizient miteinander zu kommunizieren, ein klares Ziel zu verfolgen und dabei ein ganzheitliches Verständnis von dem Produkt zu haben, trägt wesentlich zum Projekterfolg bei und sollte nicht unterschätzt werden. In diesem Kapitel wird zunächst die AOF-Methode vorgestellt und erklärt. Im weiteren Verlauf werden Epics und User Stories aufgeführt und der Unterschied und Nutzen derselben dargestellt.

5.2.1. Anwendung der AOF-Methode

Die AOF-Methode ist ein Rahmenwerk, welches dabei hilft, möglichst frühe und wesentliche Designentscheidungen zu treffen. Es gibt nichts Schlimmeres in einem Projekt als ein featuregetriebenes Design, bei dem die Designexperten ein Feature nach dem anderen in das Design einbauen, damit das Design überladen und so die Anwendung nur sehr schwer benutzbar machen. Diese Vorgehensweise resultiert meistens daraus, dass ein Mangel an den wirklich wichtigen Dingen herrscht. Anstatt dass man sich auf die wesentlichen Funktionalitäten konzentriert, versucht das Team zu viel Funktionalität einzuarbeiten. Um dieses Verhalten zu unterbinden, müssen sich Designexperten schon früh gewisse Fragen beantworten, wie z. B. „Was ist die primäre Aktivität unserer Anwendung?“, „Welche Features brauchen wir wirklich, um diese Aktivität zu unterstützen?“, und vielleicht die wichtigste Frage: „Welche Features können wir dabei auslassen?“ (Porter, 2008). Was Designexperten also brauchen, ist ein Weg, bei dem sie die vorgeschlagenen Features nach ihrem Wert für das Design priorisieren und beurteilen können. Sie müssen wissen, welche Features es wert sind, dass man Zeit und Energie dafür investiert. Dabei soll die Anwendung der AOF-Methode helfen.

Diese Methodik stützt sich auf die drei folgenden Säulen (Porter, 2008):

Fokus auf die primären Aktivitäten
Da man nun die Bedürfnisse der Benutzer bereits aus der Produktvision kennt, lässt sich daraus ableiten, warum die Benutzer die Anwendung nutzen würden. Somit sind die primären Aktivitäten bekannt.

Identifizieren der sozialen Objekte
Diese Aktivitäten, wie z. B. Lesen, Kaufen oder Bewerten, beziehen sich immer auf Objekte (Bücher, Produkte oder Bilder). Zusammengenommen kann man bereits aus diesen beiden Säulen den Kern der Anwendung sehr präzise umreißen.

Bestimmen der Kern-Features
Damit man nun von dem A und O auf die einzelnen Features gelangt, kann man aus den Verben der primären Aktivitäten bezogen auf die sozialen Objekte, mit denen die Benutzer interagieren, die ersten großen Epics und User Stories, wie z. B. „Bücher lesen“, „Bücher kaufen“ oder „Bücher bewerten“, generieren.

5.2.2. Epics und Stories - Aufbau und Unterschiede

Der ausschlaggebende Unterschied zwischen einem Epic und einer User Story besteht darin, dass ein Epic eine wesentlich allgemeiner formulierte User Story ist, die zu groß ist, um in einer einzigen Iteration implementiert zu werden. Genau aus diesem Grund muss sie in kleinere Stories aufgeteilt werden. Ein Textgerüst kann helfen, die Anforderungen an Epics und Stories hinreichend genau zu formulieren:

> **Als** ... *[Anwendertyp]*
>
> **kann/möchte ich** ... *[folgende Aktion durchführen],*
>
> **um** ... **zu** ... .*[diese Ziel zu erreichen]*

Epics

Ein Epic lässt sich dadurch unterscheiden, dass es offensichtlich aus mehreren untergeordneten Teilschritten besteht. In seiner Gesamtheit beinhaltet es aber einen vollständigen und abgeschlossenen Kundennutzen bzw. repräsentiert einen Business Value. Wichtig ist, dass ein Epic eine funktionale Anforderung beschreibt.
Ein Beispiel: „Als Kunde möchte ich ein Buch kaufen, um es lesen zu können“. Hieraus wird relativ deutlich, dass ein solcher Vorgang aus mehreren Teilschritten bestehen kann:

1. Als Kunde möchte ich ein Buch in den Warenkorb legen.
2. Als Kunde möchte ich mein Buch bezahlen können.
3. Als Kunde möchte ich eine Versandmethode auswählen können.

Dennoch bieten diese Teilschritte erst in ihrer Gesamtheit einen tatsächlichen Wert sowohl für das Unternehmen, das Bücher verkauft, als auch für den Kunden, der ein Buch kaufen möchte.

User-Stories

(Basiert auf Gesprächen mit Wolfgang Wopperer, mindmatters GmbH & Co. KG, 2011)
Die oben aufgeführten Teilschritte eines Epics sind das, was man typischerweise User Stories nennt. Auch diese besitzen ein Abgrenzungskriterium: Eine Story sollte immer eine abgeschlossene Interaktionseinheit beschreiben. Durch diesen Aspekt der Interaktion wird deutlich, dass zu einer Story auch immer ihre Benutzbarkeit durch den Endanwender gehört. Eine Funktionalität, die nicht entdeckt oder benutzt werden kann, existiert für den Benutzer nicht. Auch eine Story sollte einen funktionalen Charakter haben: „Stories“ wie „Design einbinden“ oder „etwas schön gestalten“, die sehr häufig aufkommen, weil man nicht weiß, wie man sonst den fertigen Code und das dafür vorgesehene Design zusammenbringen soll, sind nicht nur ihrer Form nach keine Stories, sondern tatsächlich auch in ihrem Wesen. Stories befinden sich auf der letzten Abstraktionsstufe vor einer, für die Umsetzung, konkreten Handlungsanweisung. Sie sind die detaillierteste Form, in der das Entwicklungsteam mit dem Product Owner über das Produkt spricht. Das Entwicklungsteam selbst wird dann in der Regel eine Story noch weiter in einzelne Tasks unterteilen, damit konkrete Implementierungsaufgaben beschrieben werden können. Man sollte dabei immer beachten, dass eine Story kein „Ticket“ ist: Sie ist nur eine Merkhilfe und ersetzt keine Diskussion mit dem Product Owner. Ganz im Gegenteil ist sie die Diskussionsgrundlage zwischen Entwickler/Designexperte und Product Owner.

Auf der anderen Seite wird häufig vergessen, dass eine Story klare Akzeptanzkriterien benötigt: „Wann ist die Interaktion angemessen abgebildet?“ Denn nur so lassen sich tatsächlich messbare Fortschritte in der Entwicklung erzielen.

Aus diesem Grund muss eine Story möglichst das INVEST-Kriterium erfüllen:

- **I**ndipendent: Eine Story ist unabhängig von anderen Stories. Eine Story darf nicht davon abhängen, dass zuerst eine andere Story umgesetzt werden muss.
- **N**egotiable: User Stories sind kein geschriebenes Gesetz. Kunden und Entwickler besprechen und präzisieren sie gemeinsam. Während der Kunde also die Funktionalität kurz und grob beschreibt, werden die Details mit den Entwicklern zusammen ausgearbeitet.
- **V**aluable: Die Stories sollten einen erkennbaren Mehrwert liefern. Ansonsten gibt es keinen Grund, sie vorzuhalten. Der beste Weg zu wertvollen Stories ist der, sie vom Nutzer selbst schreiben zu lassen.
- **E**stimatable: Eine Story muss so überschaubar sein, dass die Entwickler die Umsetzung der Anforderung abschätzen können. Zudem müssen die Entwickler natürlich über die entsprechenden fachlichen und technischen Kompetenzen verfügen.
- **S**mall: Über den konkreten Umfang von User Stories muss letztlich das Team entscheiden. Stories können „zu groß“ und „zu klein“ sein. Als grobe Regel gilt: Die komplette Umsetzung einer Story soll mindestens einen halben Personentag und maximal zehn Personentage erfordern.
- **T**estable: Die Tests bilden den Maßstab dafür, ob eine Story erfolgreich abgeschlossen wurde oder nicht. Daher muss die Testbarkeit zwingend gewährleistet sein.

5.3. Interaktive UX-Pattern-Bibliothek

In diesem Abschnitt wird ein weiteres Werkzeug vorgestellt, welches die Zusammenarbeit zwischen dem Entwickler und Designexperten während der einzelnen Sprints unterstützen soll. Es wurde bereits erwähnt, dass zwar zwei parallel verlaufende Abläufe in dem hier vorgestellten Konzept bestehen, jedoch sollten diese möglichst gleichzeitig verlaufen. Hierfür wird der interaktive Styleguide verwendet. Im ersten Abschnitt wird der Ursprung des eigentlichen Styleguides aufgeführt, außerdem wird der Nutzen sichtbar, welchen man aus dem neuen, interaktiven Styleguide ziehen kann. Im weiteren Verlauf wird etwas näher auf die Programmiersprache eingegangen, mit der dieser Styleguide erstellt wurde, und die Architektur sowie der Aufbau werden ebenfalls erläutert. Zum Schluss werden Meta-Konzepte des Styleguides vorgestellt. Hierfür werden Codebeispiele aufgeführt, und es wird erklärt, wie man gewisse Komponenten verwendet.

5.3.1. Ursprung und Nutzen des interaktiven Styleguides

Der Styleguide hat seinen Ursprung aus der Medien- und Agenturwelt. Ein Styleguide beschreibt Regeln bzw. bestimmte Elemente eines Druckerzeugnisses oder einer Webanwendung und deren Gestaltung. Diese Richtlinien sollen ein einheitliches Erscheinungsbild verschiedener Kommunikationsmittel eines Unternehmens gewährleisten. Mit diesen Richtlinien werden z. B. die einheitliche Sprache, das Layout und das Design geregelt und schriftlich in einer PDF-Datei festgehalten. Dadurch sollen Fehler bei späteren Adaptionen und Erweiterungen vermieden werden; es soll ein einheitliches Bild geschaffen werden. Es handelt sich hierbei also um ein statisches und nur schwer anpassungsfähiges Artefakt in einem sich ständig anpassenden agilen Prozess. Man kann sich vorstellen, dass ein Styleguide in der Form, wie er früher verwendet wurde, nicht gerade kommunikationsfördernd ist, geschweige denn sich schnell neuen Anforderungen anpassen lässt. Ein statischer Styleguide stellt also ein vermeidbares Hindernis in diesem Prozess dar. Die vorliegende Arbeit stellt einen Styleguide vor, der als technologische Schnittstelle zwischen dem Entwickler und dem Designexperten dienen soll, dabei die Kommunikation fördert und die Einarbeitung von Designelementen vereinfacht. Dieser interaktive Styleguide ist generisch aufgebaut und besitzt wiederverwendbare Komponenten, die aus einem richtigen, einsetzbaren Code bestehen. Dieser Code kann sofort vom Entwickler dafür verwendet werden, bestimmte Features zu realisieren und sich auf einer gemeinsamen Basis mit dem Designexperten auszutauschen. Des Weiteren gewinnt man hier den großen Vorteil, dass Elemente, die in die Anwendung eingebaut werden sollen, vorher schon auf ihre

Browser-Kompatibilität hin geprüft und Fehler im Layout beseitigt werden können, bevor sie in die eigentliche Anwendung fließen.
Somit wird die Hürde verkleinert, die Entwickler bei statischen Photoshop-Dateien oder einem statischen Styleguide nehmen müssen, um einen funktionsfähigen Frontend-Code zu produzieren. Folglich liegt die Belastung nicht am Ende eines Sprints oder eines Projektes, sondern wird kontinuierlich gering gehalten, weil man mit dem richtigen Code arbeiten kann und nicht mit statischen Bildern hantieren muss. Die Vorteile, die sich daraus ergeben, liegen auf der Hand: Man ist fähig, schon früh Fehler zu identifizieren und sie zu beseitigen; man hat die Möglichkeit, Veränderungen am Design, so oft dies benötigt wird, vorzunehmen und sie jederzeit einzubauen.
Die agilen Prinzipien fordern eine gemeinsame Verantwortung für den Code, auch bekannt als „collective code ownership“. Diese ermutigt jeden Entwickler, neue Ideen zu allen Bereichen eines Projektes beizutragen. Sie fordert, dass jeder Entwickler auch die Codefragmente der andere kennt und sie jederzeit anpassen oder korrigieren kann. Weil das ganze Team dafür verantwortlich ist, wird niemand zu einem Bottleneck für Änderungen an der Anwendung. Wenn nun „collective code ownership“ bedeutet, dass alle Entwickler für das Wohl des ganzen Codes zuständig sind, dann bedeutet „collective design ownership“, dass sowohl Designexperten als auch Entwickler Verantwortung für das erzeugte Design tragen und auch Veränderungen vornehmen können. Durch Anwendung dieses interaktiven Styleguides erreichen wir eine gemeinsame Verantwortung für das Design.

5.3.2. HAML und SASS

HAML und SASS sind Markup-Sprachen, beide vereinfachen das Erstellen von HTML- und CSS-Dateien erheblich. Während HAML entwickelt wurde, um viele Schwächen traditioneller Template-Mechanismen zu vermeiden und um die HTML-Dateien so elegant wie möglich zu gestalten, ist SASS eine CSS-Sprache, die entwickelt wurde, um den CSS-Code zu abstrahieren und dadurch einfachere CSS-Dateien zu generieren. Man hat die Möglichkeit, beide Sprachen in einem Projekt zu nutzen oder auch nur eine von ihnen. Bei HAML muss man sich z. B. nicht mehr um die schließenden Tags kümmern, weil durch Einrückung der Unterelemente diese Struktur auf natürliche Weise bewahrt wird, wobei sie für menschliche Leser viel einfacher und logischer zu erfassen ist. Der produzierte HAML-Code ist wesentlich kürzer als in der HTML-Schreibweise und lässt sich dadurch wesentlich schneller schreiben, wodurch man relativ viel Zeit spart, die man am Ende für andere, wichtigere Dinge verwenden kann. Ein weiteres wichtiges Prinzip von HAML ist, dass die Auszeichnungssprache DRY sein sollte; HAML vermeidet durch den eingerückten und klar strukturierten Code unnötige Wiederholungen von Elementen.
Im folgenden nun ein paar kleinere Code-Beispiele (HAML, 2011):

```
1  <div id=‘‘blog’’>
2      <article class=‘‘blog-entry’’>
3          <h1>Hallo das ist mein erstes Blog</h1>
4          <p>Hier werde ich ab und zu etwas reinschreiben</p>
5      </article>
6  </div>
```

Listing 5.1: HTML5 Codebeispiel für den Vergleich zwischen HTML und HAML

```
1  #blog
2      %article.blog-entry
3          %h1 Hallo das ist mein erstes Blog
4          %p Hier werde ich ab und zu etwas reinschreiben
```

Listing 5.2: HAML Codebeispiel für den Vergleich zwischen HTML und HAML

Bei HAML werden alle HTML-Tags mit vorangeführte „%“ eingeleitet, eine CSS-Klasse wird, wie bereits aus dem normalen CSS-Code bekannt, mit einem Punkt vorweg initialisiert und eine ID mit einer Raute („#“). HAML ist durch den Einsatz in Ruby-on-Rails-Anwendungen beliebt geworden, daher wird im Folgenden ein komplexeres Beispiel aus einer solchen Anwendung gegeben:

```
1  <%- @blog.comments.each do | comment | %>
2      <div class='blog-comment' id='item<%= item.id %>'>
3          <%= item.body %>
4      </div>
5  <%- end %>
```

Listing 5.3: Beispiel für ein ERB-Template einer Ruby-on-Rails Anwendung

```
1  - @blog.comments.each do | comment |
2      .blog-comment{:id => "comment_#{comment.id}"}= comment.body
```

Listing 5.4: Beispiel für ein HAML-Template einer Ruby-on-Rails Anwendung

SASS ist eine Meta-Programmiersprache und setzt auf CSS auf. Sie wird benutzt, um saubere und klar strukturierte Oberflächen zu gestalten. Im Gegensatz zu CSS bietet SASS wesentlich mehr Funktionalität an. So können Regeln verschachtelt werden, es besteht die Möglichkeit, Variablen zu benutzen, außerdem kann man sogenannte Mixins definieren, und es ist möglich, Vererbung zu benutzen. Sass bereichert also das CSS und macht es kontrollierbarer und DRY (SASS, 2011). Im Folgenden nun einige Beispiele aus den bereits genannten Bereichen — Variablen, Verschachtelung, Mixins und Vererbung.

Variablen

Durch die Fähigkeit, Variablen zu benutzen, bekommt der CSS-Code eine neue Dimension. Man hat die Möglichkeit, eine Hauptdatei zu erstellen, welche für die Farben, Schriften und Inhalte verantwortlich ist und nur aus Variablen besteht. Sie dient somit als eine Art Konfigurationsdatei für das Layout, weil Variablen aus dieser Datei in anderen, kleineren spezifischen Dateien wiederverwendet werden. Somit lassen sich globale Änderungen an einem Punkt adressieren und man erhält Kontrolle über das Aussehen der Anwendung (SASS, 2011).

CSS	SASS

`.main-navigation {`
`  border-color: #3bbfce;`
`  color: #2b9eab;`
`}`
 	
`.main-border {`
`  padding: 8px;`
`  margin: 8px;`
`  border-color: #3bbfce;`
`}`
	`$blue: #3bbfce`
	`$margin: 16px`
	
	`.main-navigation`
	`  border-color: $blue`
	`  color: darken($blue, 9%)`
	
	`.main-border`
	`  padding: $margin / 2`
	`  margin: $margin / 2`
	`  border-color: $blue`

Tabelle 5.1.: Beispiel für SASS Variablen

Verschachtelung

Durch Verschachtelung der einzelnen CSS-Klassen und IDs wird eine bessere Übersicht gewährleistet. Daraus resultiert einerseits, dass man Zeit spart, weil man nicht ständig die gleichen Klassen schreiben muss, andererseits aber werden lange und unlesbare Verkettungen von CSS-Selektoren vermieden (SASS, 2011).

CSS	SASS

`table.hl {`
`  margin: 2em 0;`
`}`
`table.hl td.ln {`
`  text-align: right;`
`}`
 	
`li {`
`  font-family: serif;`
`  font-weight: bold;`
`  font-size: 1.2em;`
`}`
	`table.hl`
	`  margin: 2em 0`
	`  td.ln`
	`  text-align: right`
	
	`li`
	`  font:`
	`  family: serif`
	`  weight: bold`
	`  size: 1.2em`

Tabelle 5.2.: Beispiel für SASS Verschachtelung

Mixins

Mixins sind eine Art von Styling-Funktion oder auch Komponenten. Sie fassen gestalterische Anforderungen zu einer Komponente zusammen, welche man dann an verschiedenen Stellen wiederverwenden kann, wenn sie benötigt werden „Sass2011“.

CSS	SASS

`#data {`
`  float: left;`
`  margin-left: 10px;`
`}`
`#data th {`
`  text-align: center;`
`  font-weight: bold;`
`}`
`#data td, #data th {`
`  padding: 2px;`
`}`
	`@mixin table-base`
	`  th`
	`  text-align: center`
	`  font-weight: bold`
	`  td, th`
	`  padding: 2px`
	
	`@mixin left($dist)`
	`  float: left`
	`  margin-left: $dist`
	
	`#data`
	`  @include left(10px)`
	`  @include table-base`

Tabelle 5.3.: Beispiel für SASS Mixins

Selektorenvererbung

Durch Vererbung der CSS-Selektoren erreicht SASS schon fast einen objektorientierten Charakter. Man ist in der Lage, gewisse Klassen durch generische Klassen zu erweitern. Somit ist man nicht gezwungen, die gleichen Sass-Definitionen wieder neu zu setzen. Außerdem sieht man auf einen Blick, von welcher Oberklasse diese Klasse direkt abgeleitet wird (SASS, 2011).

CSS

```
.error, .badError {
  border: 1px #f00;
  background: #fdd;
}

.error.intrusion,
.badError.intrusion {
  font-size: 1.3em;
  font-weight: bold;
}

.badError {
  border-width: 3px;
}
```

SASS

```
.error
  border: 1px #f00
  background: #fdd

.error.intrusion
  font-size: 1.3em
  font-weight: bold

.badError
  @extend .error
  border-width: 3px
```

Tabelle 5.4.: Beispiel für SASS Selektoren Vererbung

5.3.3. Aufbau und Architektur

Der Styleguide ist hierarchisch aufgebaut und besteht aus mehreren Stufen. Der wohl größte Unterschied ist darin zu sehen, dass es sowohl Bibliotheken als auch konkrete Implementierungen gibt. Die Bibliotheken, mit .lib.sass benannt (z. B. _tables.lib.sass), enthalten grundlegende Komponenten aus Mixins, die dann in den konkreten Dateien, z. B. tables.sass, zum Einsatz kommen. Des Weiteren gibt es einen Komponententeil, der sogenannte Widget-Styles beinhaltet für Elemente, die an unterschiedlichen Stellen der Anwendung eingebunden werden können, wie z. B. Boxen, in denen verschiedene Inhalte dargestellt werden.
Auf der höchsten Stufe befinden sich sogenannte CSS-Frameworks. Compass ist eines davon (siehe Abb. 5.2). Compass ist ein Framework, das auf der SASS-Metasprache aufbaut und dem Designexperten mit einer Vielzahl von Hilfsmethoden und Mixins hilft, den Arbeitsablauf effizienter zu gestalten. Compass ist quasi die Standardbibliothek für Sass und bietet vieles, was auch gängige CSS-Frameworks enthalten. Das Framework bietet eine große Anzahl von Mixins an, darunter Reset-Styles, Clearfixes sowie bewährte CSS-Hacks. Daneben ermöglicht Compass das automatisierte Erstellen von Sprites (CSS-Framework, 2011).

Der Schwerpunkt von Compass liegt darauf, die Anwendung der neuen CSS3-Techniken zu vereinfachen. Entwickler und Designexperten müssen sich somit keine Gedanken über die aktuelle Browserkompatibilität mehr machen oder darüber, ob sie alle Fälle abgedeckt haben, die die aktuelle — von W3C empfohlene — Schreibweise verwenden (CSS-Framework, 2011). Obwohl Compass eine sehr umfangreiche Bibliothek ist, landen im generierten CSS-Code nur die Teile, die auch tatsächlich genutzt werden. Die eigenen Stylesheets müssen also keine riesigen Dateien einbinden, sondern können punktgenau Compass-Helfer aufrufen. Richtig angewandt kommt der entstehende Code weitgehend ohne Wiederholungen aus (CSS-Framework, 2011).
Die Ebene direkt darunter ist die Konfigurationsebene (siehe Abb. 5.2). Sie besteht aus der Hauptdatei, welche die Hierarchien abbildet und die unterschiedlichen Bibliotheken und Styles aufruft. Alle Bibliotheken befinden sich in der Hierarchie darunter, die Bibliotheken sind Dateien, die spezialisierte Mixins beinhalten, welche nicht in dem Compass-Framework eingebunden sind. Diese Ebene beinhaltet Bibliotheken für Tabellen, Formulare, Typographie, Layout usw. Die darunter liegende Ebene besteht aus generischen Anwendungs-Styles; sie setzen sich aus den beiden Bereichen Typographie und Layout zusammen und dienen dem wesentlichen Erscheinungsbild der Anwendung. Die im Auslieferungszustand tiefste Ebene besteht aus generischen Elementen-Styles. Sie bauen alle auf den Bibliotheken auf und greifen auf die dort definierten Standards zu bzw. erweitern die Anwendung mit konkreten Styles für die unterschiedlichen Kontexte.
Hier findet sich nun die Ebene, auf der der Entwickler mit dem Designexperten zusammenarbeitet und das gestalterische Aussehen der Anwendung bestimmt. Es gibt allerdings noch eine Ebene darunter, die Ebene für seitenspezifische Styles. Sie sollte man möglichst vermeiden, weil ansonsten der Styleguide zu spezialisiert wird und sich dadurch nur schwer anpassen lässt. Nichtsdestotrotz kommt es durchaus vor, dass spezielle Styles für gewisse Bereiche einer Anwendung gesetzt werden müssen, weil sich nicht immer ein generischer Ansatz bewerkstelligen lässt.

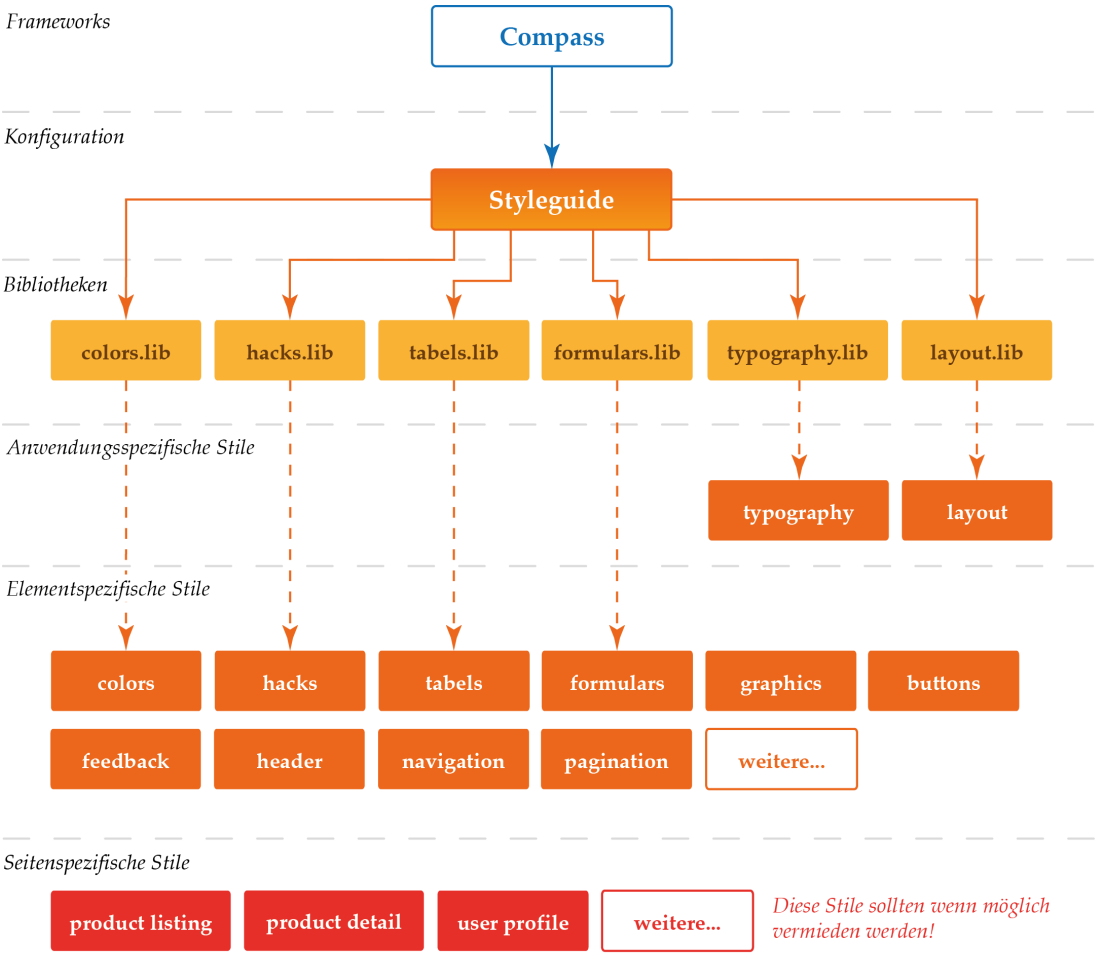


Abbildung 5.2.: Diese Grafik zeigt die einzelnen Elemente aus denen der Styleguide Zusammengesetzt ist und deren hierarchischen Aufbau

5.3.4. Meta-Konzepte im Styleguide

Man sollte während der Implementierung darauf achten, möglichst auf hoch spezifische Seiten-Styles zu verzichten. Diese erhöhen unnötig die Komplexität und damit auch den Implementierungsaufwand, außerdem sind sie für andere Entwickler oder Designexperten, die Teile des Styleguides nicht selbst gebaut haben, schwer zu verstehen und somit auch schwerer zu benutzen. Genau aus diesem Grund ist der gesamte Styleguide so generisch wie nur möglich aufgebaut und beinhaltet viele wiederverwendbare Komponenten mit klaren und aussagekräftigen Klassennamen und Elementen. Im Folgenden werden ein paar elementare Komponenten und deren Implementierung und Anwendung vorgestellt.
Der Styleguide beinhaltet in der Grundversion lediglich die wichtigsten Komponenten, die benötigt werden, um mit der Implementierung zu beginnen; er ist dafür gedacht, weiterentwickelt zu werden und mit der Webanwendung zu wachsen. Er beinhaltet eine eigene Ansicht; sobald er in einer Anwendung integriert ist, kann er über eine definierte URL erreicht werden. In dieser Ansicht können alle eingebauten Komponenten über eine eigene Navigation im Browser-Fenster erreicht und dargestellt werden. Alle gestalterischen Definitionen bezüglich des Styleguides an sich beeinflussen aufgrund eines eigenen Namensraumes (-sg) nicht die Definitionen der Anwendung. Somit kann der Styleguide selbst unabhängig von der Anwendung umgebaut werden. Es werden ein paar der grundlegenden Bausteine des Styleguides exemplarisch vorgestellt, um einen Eindruck vom Aufbau und der dahinter liegenden Meta-Konzepte zu erhalten.

Layout
Die Layoutdatei besteht hauptsächlich aus Definitionen zum Aufbau einer Seite. Hierfür wird auf die in Compass enthaltenen Mixins zurückgegriffen. Das in diesem Styleguide verwende Layout basiert auf dem Blueprint Grid-Layout (Blu, 2011). Ein gesamtes Seitenlayout besteht aus 24 Spalten, dem sogenannten Grid. Diese Spalten sind wiederum in zwei Blöcke unterteilt. Der erste Block gibt die Breite einer Spalte an, der zweite den Abstand zur nächsten Spalte (24 Spalten insgesamt, davon 20px für Breite und 20px für Abstand, dies entspricht einer Gesamtbreite von 24-mal 40px, also 960px). Diese 960px haben sich als Standard für Webanwendungen durchgesetzt, weil sie sich optimal auf den gängigen Monitoren und deren Auflösungen anzeigen lassen (Blu, 2011).

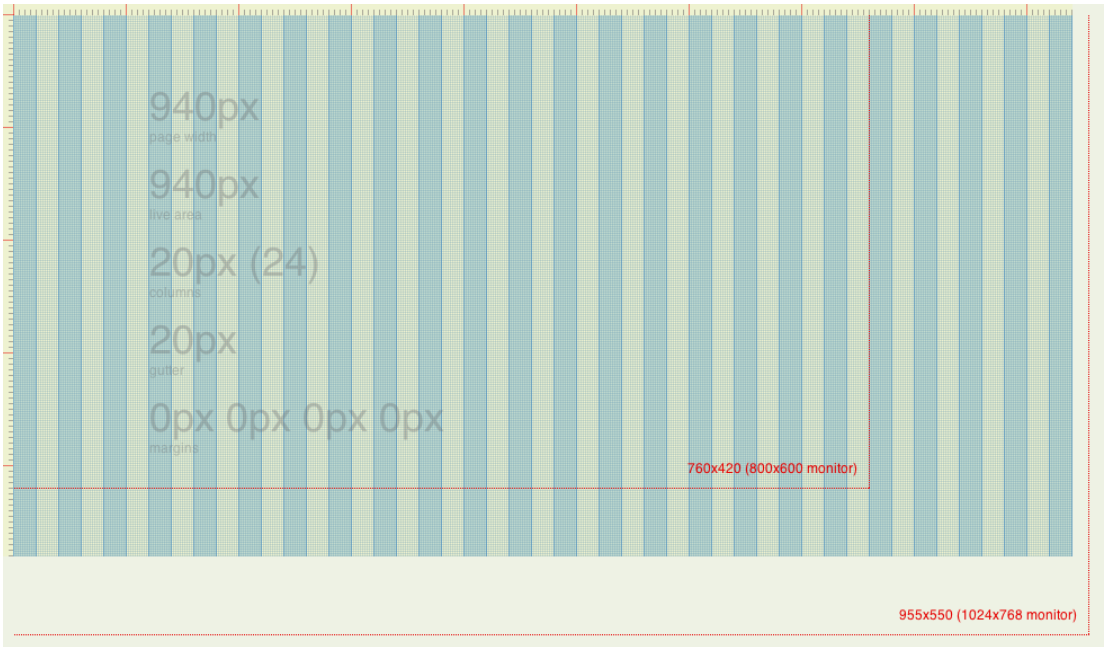


Abbildung 5.3.: Beispiel eines Standard Blueprint Grid-Layout

SASS Beispiel für das Blueprint Grid-Layout

```
#container
  +container
/* Das Mixin (+container) setzt sich aus den folgenden Werten zusammen */
/* 24 Spalten insgesamt */
$blueprint-grid-columns: 24
/* Spaltenbreite */
$blueprint-grid-width: 20px
/* Abstand zwischen den Spalten */
$blueprint-grid-margin: 20px

/* Gesamtbreite einer Spalte im Grid */
$blueprint-grid-outer-width:
$blueprint-grid-width + $blueprint-grid-margin

/* Gesamtbreite des Container Elements */
$blueprint-container-size:
$blueprint-grid-outer-width * $blueprint-grid-columns - $blueprint-grid-
    margin
```

Listing 5.5: SASS Definition des Container Elementes (siehe auch Abb. 5.3)

Im Folgenden sieht man eine Standarddefinition eines zweispaltigen Layouts; die linke Seite ist für den Inhalt, die rechte ist eine Seitenspalte für zusätzliche Inhalte, wie von gängigen Internetseiten oder Anwendungen bekannt.

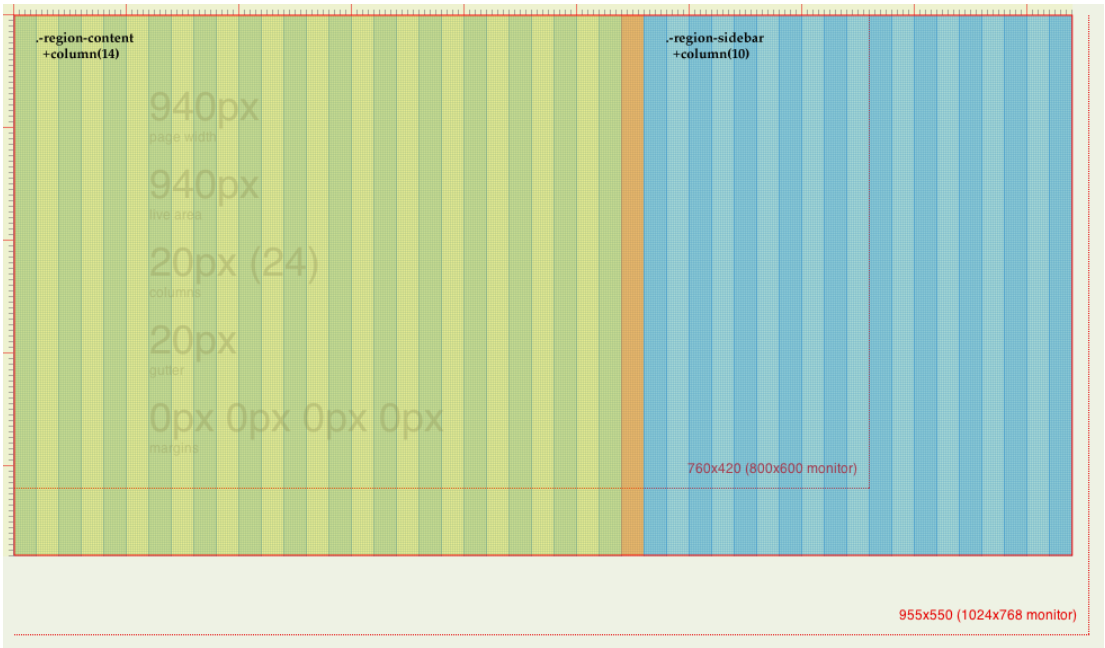


Abbildung 5.4.: Aufbau eines Beispiel Layouts

```
/* Die CSS-Klasse für das Basis-Layout der Anwendung */
.-layout-base
  /* Es bekommt eine Spaltenbreite von 24 zugewiesen */
  +column(24)

  /* Davon gehören 14 zum eigentlichen Inhalt */
  .-region-content
    +column(14)
  /* 10 Spalten sind für die Seitenleiste reserviert, mit ''true'' gibt man
       an das es sich um die letzte Spalte im Layout handelt, dadurch wird
      der Abstand von 20px ($blueprint-grid-margin) von dem Element
      abgezogen, die reele Gesamtbreite ist somit 940px */
  .-region-sidebar
    +column(10, true)
```

Listing 5.6: SASS-Code zum Beispiel Layout (siehe Abb. 5.4)

Die Layoutbibliothek kann nun spezialisierte Lösungen für bestimmte Anwendungsfälle beinhalten, z. B. ein Grid-Layout für Spalten mit prozentualer Verteilung, z. B. 50 % für je ein Container, anstelle einer ganzen Zahl (12 Spalten für ein Container). Es können aber bezogen auf das Layout der Anwendung auch andere Ansätze in dieser Datei realisiert werden.

Formulare

Formulare stellen in Webanwendungen einen der wichtigsten Bausteine dar. Gleichzeitig sind sie in ihrer Funktionalität sehr komplex und stellen besondere Herausforderungen bei ihrer Implementierung dar. In vielen Webanwendungen verhalten sich Formulare nicht konsistent über die gesamten Anwendungen hinweg. Oft wird auch die Anzeige und Validierung von Fehlern vernachlässigt, so dass der Benutzer schnell genervt und überfordert ist. Durch die Formularkomponente im Styleguide wird versucht, ein anwendungsübergreifendes Formular-Framework zu etablieren, welches über die gesamte Anwendung konsistent aufgebaut ist und dessen Verhalten und Handhabung der Benutzer so schnell erlernt.
Der Styleguide bietet mit den entsprechenden Sass-Dateien nicht nur eine Möglichkeit, das Layout der Formulare zu bestimmen, es existiert zugleich ein Rails-Helper, der sich um das Verhalten im Falle eines Eingabe- oder Validierungsfehlers kümmert. Der Helper baut bei vorhandenen Fehlern eine eigene Struktur auf und setzt Fehlerklassen an bestimmte Elemente. Folglich erhält das Formular durch die Style-Dateien ein explizites Aussehen.

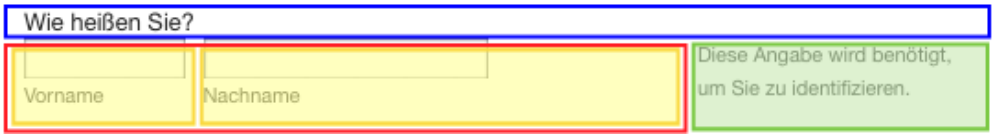


Abbildung 5.5.: Aufbau und Anordnung eines normalen Formulars

```
%form.standard-rows
  %ol
    /* Jedes List-Element ist ein Form Step */
    %li
      /* Dieser Step besteht aus einem Titel, blau gekennzeichnet */
      .step-title Wie heißen Sie?

      /* Und einem Inhalt, rot gekennzeichnet */
      .step-content

        /* Dieser ist wiederum in unterschritte eingeteilt, gelb
            gekennzeichnet */
```

```
12          .substep
13            %input{:type => "text", :size => 10}
14            %label Vorname
15          .substep
16            %input{:type => "text"}
17            %label Nachname
18        /* Es kann optional ein Hinweis für diesen Step erstellt werden, grün
               gekennzeichnet */
19        .step-hint
20          %p Diese Angabe wird benötigt, um Sie zu identifizieren.
```

Listing 5.7: HAML Beispiel für ein Standard Formular (siehe Abb. 5.5)

```
1  form.standard-base
2  /* Diese beiden Mixins sind in der Forms-Bibliothek definiert, sie
       beinhalten Meta-Styles für die Typographie und das Layout eines
       Standard Formular */
3    +standard-base-typo(lighten($feedback-negative, 67%))
4    +standard-base-layout
5
6  /* Ein Standard Formular, dieser erbt von der Ober-Klasse .standard-base
       und wird um bestimmte Breiten für den Inhalt erweitert, dies ist ein
       Anwendungsspezifischer Wert und kann durchaus abweichen */
7  form.standard-rows
8    @extend form.standard-base
9    .step-content, .subglobal-errors
10     width: 367px
11   .step-hint
12     width: 143px
```

Listing 5.8: SASS Beispiel für ein Standard Formular (siehe Abb. 5.5)

Hat man erst einmal ein Standardlayout für Formulare aufgebaut, lassen sich Änderungen relativ leicht realisieren. Im Standardlayout werden die Bezeichner jedes einzelnen Feldes oberhalb des Eingabefeldes angezeigt. In dem folgenden Beispiel wird nun das globale Verhalten des Formulars verändern, indem zwei neue Klassen hinzugefügt werden (.standard-columns-left und .standard-columns-right). Erstere richtet die Bezeichner links neben dem Eingabefeld aus, letztere erweitert das Verhalten und setzt die Bezeichner zwar links neben die Felder, jedoch werden diese rechtsbündig ausgerichtet.

Erweiterung des Standardverhaltens

Abbildung 5.6.: Formular Elemente mit nach links angeordneten Bezeichnungen

```
/* Verhalten dieser Klasse wird durch den eigenen Namen deutlich gemacht */
form.standard-columns-left
  /* Diese Klasse erbt von der Klasse .standard-rows, schwarz
      gekennzeichnet */
  @extend form.standard-rows
  +padded-column(19, 20px)
  /* Der Titel, blau gekennzeichnet */
  .step-title
    +padded-column(5, 10px)
  /* Der Inhalt, rot gekennzeichnet */
  .step-content
    +column(8)
```

Listing 5.9: Formular Erweiterung um Links ausgerichtete Feldbezeichner (siehe Abb. 5.6)

Abbildung 5.7.: Formular Elemente mit nach rechts angeordneten Bezeichnungen

```
/* Diese Klasse verändert das Verhalten der .standard-column-left klasse um
    die nach rechts ausgerichteten Titel sie Abb. 5.8 */
form.standard-columns-right
  @extend form.standard-columns-left
  /* Blau gekennzeichnet */
  .step-title
    text-align: right
```

Listing 5.10: Formular Erweiterung um Rechts ausgerichtete Feldbezeichner (siehe Abb. 5.7)

Folglich hat man die Möglichkeit nur durch Verwendung der unterschiedlichen Klassen, das gesamte Verhalten eines Formulars zu steuern und zu verändern, und das konsistent über die ganze Anwendung hinweg.

Buttons

Ein weiterer wichtiger Bestandteil einer Webanwendung sind die Buttons. Sie tauchen an unterschiedlichen Stellen einer Anwendung auf und es können unterschiedliche Funktionen mit ihnen aufgerufen werden. Man kann Buttons auf drei verschiedene Arten erzeugen: als Link-Element, als Input-Element oder als reines Button-Element. Diese unterscheiden sich nicht nur hinsichtlich es jeweils gegebenen Anwendungsfalles, sondern zudem in ihrem Aussehen. Genau hier liegt das Problem: Es ist relativ schwer und mühsam, das Aussehen dieser drei Elemente konsistent und browserübergreifend zu implementieren.
Um diese Problem lösen zu können, wird ein Ruby-Gem namens „Fancy-Buttons“ verwendet. Dieses Gem liefert browserübergreifend funktionierende und gleich aussehende Buttons aus. Weil es auch in Sass und Ruby geschrieben ist, lässt es sich auf eine Vielzahl von Frameworks adaptieren. Entsprechend wird es als Grundlage für den Styleguide verwendet, jedoch ist es durchaus möglich, ein eigenes Button-Framework zu implementieren.

```
1 /* Dies sind drei Möglichkeiten einen Button zu implementieren, in der
      Grundinstallation des Styleguides gibt es drei unterschiedlichen Arten
      von Buttons - Standard (.btn-standard), Call-to-action (.btn-cta) und
      recommended (.btn-recommended) */
2 %button.btn-standard Standard
3 %a.btn-standard{:href => "#"} Standard
4 %input.btn-standard{:type => "submit", :value => "Standard"}
5
6 /* Ausserdem gibt es noch ein verhalten wenn der Button vom Benutzer nicht
      betätigt werden kann (.disabled) */
7 %button.btn-standard.disabled Standard
8 %a.btn-standard.disabled{:href => "#"} Standard
9 %input.btn-standard.disabled{:type => "submit", :value => "Standard"}
```

Listing 5.11: HAML Beispiel für Buttons

```
1 /* Das fancy-button Mixin aus dem bereits oben erwähnten Gem liefert
      verschiedene Konfigurationsmöglichkeiten um die Struktur und die Farben
       eines Buttons anzupassen und zu verändern */
2 .btn-standard
3 /* Mögliche Parameter für dieses Mixin sind - Farbe, Schriftgröße, Radius
      der Ecken und Randstärke */
4   +fancy-button($light-grey, $font-size, $font-size)
5   +font-family
```

```
/* Dies sind speziel für den Internet-Explorer definierte Hacks um eine
    korrekte und Browser übergreifende Anzeige der Elemente zu
    gewährleisten */
button.btn-standard, input.btn-standard
  #padding-top: 0
  #padding-bottom: 0
  #position: relative
  #top: -1px
```

Listing 5.12: SASS Definitionen für Buttons

Man erhält damit die Möglichkeit, nur durch Verwendung unterschiedlicher Klassen das Verhalten und Aussehen der Elemente zu verändern. Es sollte jedoch darauf geachtet werden, dass aussagekräftige Namen wie z. B. „btn-cta disabled“ verwendet werden. So wissen der Entwickler und der Designexperte auf einen Blick, wofür dieses Element zuständig ist und wie es sich gerade verhält oder verhalten soll.

Komponenten/Widgets

Dieser Bereich des Styleguides ist für Komponenten oder Widgets einer Anwendung gedacht, die an mehreren Stellen in einer Webanwendung zum Einsatz kommt. Komponenten können z. B. Behälter für bestimmten Inhalt sein, die ein wiederkehrendes und einheitliches Aussehen und Verhalten haben sollen, unabhängig davon, an welcher Stelle sie in der Anwendung eingesetzt werden. Widgets hingegen sind z. B. Autocomplete-Eingabefelder. Diese einmal gebauten Elemente können dann einfach an verschiedenen Orten platziert werden.

Farben

Ein wichtiger Bereich sind auch die Farben einer Anwendung. Diese werden in einer Farbbibliothek festgehalten und in der Styleguide-Ansicht ausgegeben. So kann der Designexperte jederzeit sehen, wie sich die von ihm ausgewählte Farbe im Web-Browser verhält. Außerdem sind die Farben bestimmten Variablen im Sass-Code zugewiesen, so dass auch der Entwickler erkennen kann, welchen Zweck diese Farben erfüllen sollen und wie er auf welche dieser Farben zugreifen kann. Es wird generell zwischen Primär- und Sekundär- sowie Feedback- und Action-Farben unterschieden.

```
/* Hier werden die Farb-Variablen ‘$red’ & ‘$green’ gesetzt */
$red: #820000
$green: #467302
```

Listing 5.13: SASS Definitionen für Farben

Typographie

In diesem Segment werden alle Formatierungen vorgenommen die mit der Typographie zu tun hat. Es gibt eine Auflistung aller Überschrift-Typen sowie verschiedenartige Listen (Geordnet, Ungeordnet usw.). Diese Mixins werden folgendermaßen eingesetzt, somit hat man jederzeit die Möglichkeit ein spezielles Verhalten hinzuzufügen ohne das generelle Aussehen der Überschriften zu verändern.

```
/* Hier werden Mixins für Überschriften definiert */
=h1
  +font-metrics(32px, 2 * $line-height)
  margin-bottom: $line-height
  font-weight: bold

=h2
  +font-metrics(24px, 3 * $line-height/2)
  margin-bottom: $line-height
  font-weight: bold
h1
  +h1

h2
  +h2
```

Listing 5.14: SASS Definitionen für Typographie-Mixins

In der Typographischen Bibliothek können sowohl verschiedene Schriftgrößen als auch unterschiedliche Schriftarten oder jegliche anderen Änderungen im Schriftbild unternommen werden.

```
/* Das Mixin für die default Schriftart, über die beiden Mixins werden die
    Art und Größe der default Schrift definiert */
=typo-default
  +font-family
  +font-metrics($font-size, $line-height)
  color: $font-color
  font-weight: normal
  text-decoration: none
```

Listing 5.15: SASS Definitionen für typographische Grundeinstellung

Des Weiteren werden hier auch alle Typographischen HTML-Elemente und deren Verhalten zurückgesetzt damit eine einheitliche Umgebung geschaffen wird, auf der man dann einfache Browserübergreifende gestalterische Formatierungen vornehmen kann.

SASS Beispiel

```
=global-reset
  html, body, div, span, applet, object, iframe,
  h1, h2, h3, h4, h5, h6, p, blockquote, pre,
  a, abbr, acronym, address, big, cite, code,
  del, dfn, em, font, img, ins, kbd, q, s, samp,
  small, strike, strong, sub, sup, tt, var,
  dl, dt, dd, ol, ul, li,
  fieldset, form, label, legend,
  table, caption, tbody, tfoot, thead, tr, th, td
    +reset-box-model
    +reset-font
  body
    +reset-body
  ol, ul
    +reset-list-style
  caption, th, td
    +reset-table-cell
  a img
    +reset-image-anchor-border
```

Listing 5.16: SASS Definitionen zum globalen zurücksetzen aller Standarddefinitionen der Browser

Darüber hinaus wird in diesem Bereich das Verhalten und das Aussehen von Links definiert, wie im folgenden Beispiel zu sehen ist:

```
a
  color: $link-color
  &:hover, &.hover
    color: $link-hover-color
  &:active, &.active
    color: $link-active-color
```

Listing 5.17: SASS Definitionen über das Verhalten von Hyperlinks

Dies sind die wesentlichen Elemente, die zur Grundausstattung des Styleguides gehören. Der Styleguide beinhaltet noch weitere Bereiche wie Tabellendefinitionen und Werbebanner-Widgets. Außerdem kann und soll diese Grundausstattung jederzeit um weitere Bereiche erweitert werden. Insofern vermitteln die genannten Elemente nur einen groben Eindruck, der Umfang des Styleguides ist wesentlich größer. Um einen Überblick über dessen Inhalt zu bekommen, sollte man sich den Quellcode anschauen, der dieser Arbeit beigefügt ist. Ein wichtiger Faktor dieses Styleguides ist, dass er die beiden unterschiedlichen Sichten der Entwickler und der Designexperten bedient. Er ermöglicht es diesen beiden Parteien, auf einer gemeinsamen Basis zu arbeiten und somit ein gemeinsames Verständnis aufzubauen. Er bedient auf der einen Seite die technische Sicht des Entwicklers, indem er es ihm ermöglicht, Komponenten und Strukturen während der Implementierung wiederzuverwenden bzw. anzuwenden. Auf der anderen Seite bedient er die gestalterische Sicht des Designexperten, indem er es erlaubt, die gestalterischen Elemente direkt auf dem Bildschirm zu sehen. Er regt dazu an, zu kooperieren und gemeinsame Lösungswege für bestehende Probleme zu erarbeiten. Er ist die Grundlage zum Pair Designing und unterstützt folglich das inkrementelle Design.

5.4. Paperprototyping als Kommunikationsschnittstelle

Paper Prototyping[1] ist eine schnelle, effiziente und preiswerte Methode, um Benutzeroberflächen ohne Programmieraufwand zu modellieren und mit echten Benutzern zu testen. Eine Anwendung zu schreiben oder eine Internetseite zu entwerfen ist verhältnismäßig kompliziert, beim Paper Prototyping wird die Oberfläche der Anwendung zuerst auf einem Blatt Papier realisiert, bevor sie überhaupt implementiert wird. Durch Paper Prototyping lassen sich sowohl Usability Tests als auch Design Tests unter realistischen Bedingungen durchführen. Dabei werden die Oberflächen von einer Person manipuliert, die den Computer spielt; sie gibt der Testperson dabei keinerlei Hinweise auf die Funktionsweise der Oberfläche (Obexer, 2007).
Paper Prototyping dient vielen Intentionen. Während unterschiedlichen Konzeptionsphasen kann es zum Brainstorming eingesetzt werden, um Ideen zu sammeln und auf Papier zu bringen. Schritt für Schritt entstehen somit die Oberflächen, so dass die Erwartungen und Vorstellungen aller Teammitglieder vereint werden. Steht schließlich das Interaktionsdesign, kann man es anhand von Paper Prototyping auf seine Benutzbarkeit hin testen. In einem solchen Test mit echten Benutzern interagieren diese mit dem Prototypen, als sei er ein reeller Computer. Letzterer wird durch eine andere Person simuliert, die entsprechend der Eingaben des Benutzers die Oberfläche modifiziert. Genauso wie bei echten Usability Tests mit einer fertigen Anwendung können auch hier mögliche Barrieren identifiziert werden, bevor mit der eigentlichen Implementierung begonnen wird (Snyder, 2003a) (Obexer, 2007).
Paper Prototyping liefert jedoch keine Storyboards, bei denen eine Serie von Zeichnungen die Oberfläche aufzeigt, die benötigt wird, um eine bestimmte Aufgabe auszuführen. Ebenso sind sie keine Wireframes, die ein bestimmtes Layout oder eine bestimmte Navigation definieren oder aufzeigen, an welcher Stelle ein bestimmter Inhalt platziert wird. Außerdem sind sie keine Kompositionen, die das konkrete Aussehen wie Farben, Schriften oder Aufbau zeigen (Snyder, 2003a).

[1] Interessanter Blog Artikel über das PaperPrototyping von „A List Apart“, url: http://www.alistapart.com/articles/paperprototyping/, gesehen am: 18.12.2011

5.4.1. Sinnvolle Anwendung von Paper Prototyping

Wann ist es sinnvoll, auf das Paper Prototyping zurückzugreifen?(Snyder, 2003b)

- **Es gibt viele unterschiedliche Vorstellungen bezüglich des Designs.**
  Unter Umständen kann es sein, dass die Gestaltung des Produktes vom Endbenutzer überhaupt nicht verstanden wird oder mehrere Ideen und Vorstellungen über die Oberfläche unter den Projektbeteiligten herrschen. Hier besteht die Möglichkeit, mit einer kleinen Gruppe von Benutzern herauszufinden, welcher der unterschiedlichen Entwürfe der sinnvollste und zielführendste ist.

- **Es gibt verschiedene Artefakte des Produktes, die noch unklar sind.**
  Im Projektverlauf tauchen immer wieder Momente auf, bei denen man in Versuchung kommt, mit den Teilen zu beginnen, die einfach und verständlich sind. Dadurch kann es zu Problemen kommen, weil schwierigere Teile zunächst zeitlich nach hinten geschoben werden, die dann aber im Nachhinein nicht mehr ohne Weiteres lösbar sind.

- **Das Verhalten von Benutzern soll sich verändern.**
  Beim Gestalten der Oberflächen zum Zwecke der Einfachheit und Benutzbarkeit für den Benutzer kann es vorkommen, dass nach der Vereinfachung der Benutzer die Oberflächen nicht mehr versteht oder akzeptieren möchte.

- **Konzepte oder Terminologien sind für den Benutzer neu.**
  Ein wesentlicher Aspekt ist, ob Benutzer die Konzepte, die in einem Interface herrschen, verstehen. Weil die Technologien sich schneller entwickeln als die menschliche Sprache, muss man neue Begrifflichkeiten einführen, um diese zu beschreiben und den Benutzer nicht zu verwirren.

- **Man erwischt sich selbst dabei, ein bestimmtes Design vor anderen zu verteidigen.**
  Wenn Designexperten eine Oberfläche präsentieren und von der anderen Seite Gegenargumente kommen, sollten sie nicht anfangen, ihre Oberfläche zu verteidigen. Papierprototypen eignen sich sehr gut dafür, Oberflächenfehler aufzudecken.

5.4.2. Durchführung von Paperprototyping

Am Anfang stellt der Moderator das Team vor, welches den Test durchführen wird. Danach stellt er die Aufgabe vor, die bei diesem Test absolviert werden soll, und die zu erreichenden Ziele. Der eigentliche Ablauf eines solchen Tests verläuft interaktiv und wird von einem Moderator durchgeführt. Er fordert die Testperson zunächst auf, den Papierprototypen wie eine herkömmliche Anwendung zu benutzen und bestimmte Aufgaben zu absolvieren, wie z. B. „Versuchen Sie, sich einzuloggen“ oder „Suchen Sie nach einem bestimmten Modell Schuhe im Katalog dieser Webseite“. Dies sind typische Aufgaben in einen solchen Test Task (Maaß u. Savova, 2011).
Neben dem Moderator und dem Benutzer spielt eine weitere Person — die in den Prozess der Erstellung des Prototypen involviert war — den „menschlichen Computer“ und reagiert auf die Eingaben des Benutzers, indem sie die von dem Benutzer „besuchten“ Seiten zusammenstellt und „den Bildschirm“ anpasst. Sollte es dabei zu einer Situation kommen, in der der „menschliche Computer“ keine angemessene Reaktion geben kann (z. B. weil ihm die entsprechende Seite oder ein Button fehlt), muss eine Lösung schnell skizziert und der Test fortgesetzt werden. Während des Tests werden die Benutzer zum lauten Denken motiviert, um Zweifel und Unklarheiten beim Umgang mit dem Prototypen aufzudecken. In diesem Zusammenhang gilt es jedoch auch darauf zu achten, keine gezielten Anweisungen oder Wertungen der Vorgehensweise zu erteilen. Der Moderator darf nur eine beobachtende Rolle einnehmen. Auf diese Weise kann er erkunden, ob die Benutzer den Zweck und den Aufbau der Anwendung verstanden haben und wo eventuelle Verständnis- und Bedienbarkeitsprobleme bestehen (Maaß u. Savova, 2011).
Die Handlungen und Reaktionen der Benutzer gilt es während des gesamten Tests zu protokollieren. Diese Dokumentation ermöglicht eine spätere Auswertung. Bei wiederkehrenden Problemen und Hinweisen seitens der Benutzer, z. B. in Bezug auf unverständliche Beschriftungen, fehlende Buttons oder Eingabefelder, wird der Prototyp für alle folgenden Tests angepasst, wobei in jeder Testrunde im Idealfall etwa drei bis vier Benutzer einbezogen werden. Die Auswertung der Tests bildet die Grundlage, um die konkreten Benutzeranforderungen abzuleiten und zu verifizieren. Diese müssen dann später bei der Erstellung der Endversion berücksichtigt werden. An der Durchführung eines solchen Tests ist das gesamte Team beteiligt, sowohl Designexperten und Entwickler als auch Techniker und Nichttechniker, wobei sich die Techniker und Nichttechniker die Waage halten sollten (Maaß u. Savova, 2011).

Folgende Rollen nehmen an der Durchführung des Tests teil:

- Benutzer
  Eine Person die repräsentativ für die Zielgruppe des Benutzer Tests ist
- Computer
  Eine Person die während des Tests das Papier verändert um das Verhalten der Anwendung bei der Benutzung zu simulieren. Diese Person gibt keine Hilfestellungen oder Erklärungen zur Funktionsweise der Anwendung ab.
- Moderator
  Eine Person die in Benutzbarkeit von Software geschult ist oder sich damit auskennt um die Durchführung des Benutzertests zu leiten
- Beobachter
  Eine Person die nicht in den Test involviert ist und lediglich Notsitzen bei der Durchführung macht. Dieser Person ist nur ein stiller Beobachter.

Usability Test Task
Ein Usability Test Task basiert auf einem Bedürfnis des zu testenden Profils. Dabei deckt er Fragen ab, die wichtig sind für den Erfolg des Produkts und der dahinter liegenden Geschäftsidee. Er sollte möglichst nicht zu allgemein formuliert sein, aber auch nicht so speziell, dass der Benutzer überfordert ist. Aufgrund dessen sollte er eine endliche und vorhersagbare Anzahl von möglichen Lösungen beinhalten und keine unlösbare Aufgabe für den Benutzer darstellen; ein klarer Endpunkt sollte für den Benutzer erkennbar sein. Außerdem sollte der Task zur Interaktion anregen, nicht zur Meinungsäußerung. Die Praxis hat gezeigt, dass ein solcher Task verschwendete Zeit ist, solange diese Anforderungen nicht erfüllt sind.

Bestandteile eines Usability Tests sind die folgenden:

- Die Ziele und Zielgruppe muss vorher definiert werden
- Die Aufgaben für die Zielgruppe müssen erstellt werden
- Die nötigen Papierprototypen müssen erstellt werden
- Man sollte den Test vorher selbst durchgespielt haben
- Testen und nach jedem Test verbessern
- Die Erkenntnisse müssen festhalten und kommuniziert werden

5.4.3. Vorteile und Nachteile von Paperprototyping

Im folgenden sind die Vor- und Nachteile aufgelistet, die bisher durch das anwenden eines Paper Prototyping identifiziert werden konnten:

Vorteile	Nachteile
Frühes Feedback für Entwickler und Designexperten	Komplexe und filigrane grafische Interaktionen schlecht Simulierbar
Verbessert die gemeinsame Sicht auf das zu Grunde liegende Produkt	Es wird kein lauffähiger Code produziert
Schnelle Änderungen der Oberflächen ohne Implementierungsaufwand	Es werden nicht alle Probleme auf der Oberfläche aufgedeckt
Verbessert die Kommunikation mit allen Projektbeteiligten, insbesondere zwischen dem Kunden und Designexperten	In kurzen Projekten erzeugt das zu viel Overhead
Es werden keine technischen Vorkenntnisse benötig	Veraltete schnell während der Projektlaufzeit
Interaktionsmodelle in der Anwendung werden Sichtbar	Kann von Entwicklern als minderwertige Arbeit wahrgenommen werden
Testbare Interaktionen werden bereits vor der Implementierung erstellt	technische Vorgänge nicht immer nachvollziehbar
Möglichkeit die Abfolge ganzer Dialogsequenzen direkt zu variieren	
Anwendbarkeit des Interaktionsdesigns kann getestet werden	
Frühe Erkennung mögliche Schwachstellen identifiziert	
Definition einer gemeinsamen Sprache zwischen Entwicklern, Designexperten und Stakeholdern	

Tabelle 5.5.: Vorteile und Nachteile von Paperprototyping

6. Fallstudie

In diesem Kapitel wird die durchgeführte Fallstudie zum agilen Design beschrieben. Zunächst werden die einzelnen Rollen benannt, die an dieser Fallstudie teilgenommen haben, und deren Ziele auf ein Projekt bezogen. Des Weiteren werden die Aufgaben in einem agilen Projekt und konkrete Schnittstellen zwischen diesen Rollen benannt. Der nächste Abschnitt zeigt dann den Fragebogen zu dieser Fallstudie. Es werden alle Fragen in einer kategorischen Anordnung aufgezeigt. Zum Schluss werden die Ergebnisse der Fallstudie ausgewertet und dabei konkrete Probleme erkannt sowie deren Relevanz abgewogen.

6.1. Einleitung

Die für die vorliegende Arbeit durchgeführte Fallstudie wird dafür verwendet, die in dieser Arbeit vorgestellten Lösungsansätze zu bekräftigen. Zusätzlich sollen evtl. noch weitere Schwachpunkte bei der Integration eines agilen Designs identifiziert werden.
Die Fallstudie wurde ausschließlich im agilen Umfeld durchgeführt; befragt wurden die Projektbeteiligten, die in einem Projekt involviert waren, bei dem das agile Design angewandt wurde. Der Fragebogen zu dieser Fallstudie beinhaltet nicht nur Fragen konkret zu solch einem Projekt, sondern bezieht sich allgemein auch auf andere Projekte, die die Teilnehmer durchgeführt oder an denen sie beteiligt waren. Dabei ergeben sich für diese Fallstudie die im agilen Kontext gängigen Rollen wie Product Owner, Scrum Master, Entwickler und Designexperte. Diese Rollen haben ein gemeinsames Ziel — sie wollen ein erfolgreiches und qualitativ hochwertiges Produkt abliefern. Es gibt aber auch explizite Ziele, die zwar am Ende zu dem gemeinsamen Ziel führen, jedoch während der Zusammenarbeit explizit von jedem einzelnen überprüft werden müssen. Die Aufgabe der Fallstudie ist es, mögliche kritische Punkte im Prozess oder bei der Zusammenarbeit zu identifizierten.
Dafür wurden diverse Fragen in den Bereichen Allgemeines, Zusammenarbeit, Prozess und Tools (Produktvision, HTML-Styleguide und Paper Prototyping) gestellt. Es wurde einerseits versucht, das vorhandene Verständnis für die Problematik zu erfragen, andererseits zielten die Fragen darauf ab, wie mit den gegebenen Problemen umgegangen wurde. Die Art der Fallstudie ist eine Problemfindungsfallstudie. Das zugrunde liegende Problem wurde ausdrücklich benannt und war den Teilnehmern auch bekannt. Es geht nun darum, diese Probleme in ihrer Relevanz abzuwägen.

6.2. Fragebogen zur Fallstudie über „Agile Design“

Die Beantwortung des Fragebogens dauert im Durchschnitt ca. 20 Minuten. Die hier aufgeführten Fragen sollen lediglich einen groben Überblick über den Fragenkatalog geben. Der Fragebogen beinhaltet verschiedene Typen von Fragen und deren möglichen Antworten. Im folgenden werden nun diese Typen kurz dargestellt. Der gesamte Fragenkatalog kann im Anhang A eingesehen werden.

- **Typ 1**
  Wie lange haben Sie schon Erfahrungen in agilen Prozessen?
  Mögliche Antworten: weniger als ein Jahr, 2-3 Jahre, 4-5 Jahre, mehr als 5 Jahre, weiß ich nicht

- **Typ 2**
  Haben Sie vor Ihrem letzten Projekt schonmal vom agilen Design gehört?
  Mögliche Antworten: Ja, Nein

- **Typ 3**
  Wer ist für das aufrechterhalten der Kommunikation während eines agilen Projektes verantwortlich?
  Mögliche Antworten: Product Owner, Scrum Master, Scrum Team, Alle, Weiß ich nicht

- **Typ 4**
  „Fehlende Kommunikation im Projekt trägt zu einer schlechteren Qualität des Produktes bei."
  Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht

- **Typ 5**
  Wie war die Zusammenarbeit zwischen Design und Entwicklung?
  Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht

- **Typ 6**
  Ist solches technisches Wissen der Designer ihrer Erfahrung nach vorhanden?
  Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht

- **Typ 7**
  Wie wurde mit neuen Features oder der Neupriorisierung von bestehenden Features im Prozess umgegangen?
  Beantwortung in ein oder zwei Sätzen oder durch Angabe bestimmter Schlagwörter

6.3. Auswertung der Ergebnisse

Im Folgenden werden nur die relevantesten Ergebnisse der Auswertung aufgeführt. Die gesamte Auswertung der Fallstudie inklusive einer grafischen Darstellung der Ergebnisse können dem Anhang B entnommen werden.
An dem Fragebogen nahmen insgesamt 16 Personen teil. Darunter waren ein Scrum Master, drei Product Owner, insgesamt neun Entwickler, zwei Designexperten und ein Konzeptionierer. Die überwiegende Mehrheit der Befragten hatte im Durchschnitt mehr als drei Jahre Erfahrung mit agilen Methoden. Die Befragten wurden nach Schlagwörtern gefragt, die ihrer Meinung nach die agile Entwicklung ausmachen. Dabei sind die fünf am meisten

genannten Begriffe Kommunikation, Feedback, Flexibilität, Mut und Einfachheit; vier dieser Begriffe sind konkrete Werte agiler Methoden.
Agiles Design als solches war 75 % der Personen bekannt, hier spiegelt sich entsprechend auch eine gewisse Erfahrung wider. Aufgrund dieser Erfahrung ist es nicht überraschend, dass fast alle Befragten einen externen Designexperten als Risikofaktor in einem agilen Projekt einstufen. Dabei ist ihrer Ansicht nach die gravierendste Auswirkung in der erschwerten Kommunikation zu sehen, die mit wesentlich mehr Koordinationsaufwand verbunden ist. Ein Drittel nannte den auseinanderlaufenden Informationsfluss als weitere Auswirkung. Dieser beeinflusst nicht nur die Projektlaufzeit, oft ist dadurch die Verantwortung der aktuellen Rollen den Personen nicht bekannt, so dass sie nicht wissen, wie sie handeln sollen, oder es ihnen schwerfällt, Entscheidungen zu fällen. Zudem ergibt sich im weiteren Verlauf ein wesentlich höherer Aufwand bei der Implementierung.
Diese noch recht allgemeine Einschätzung zeigt bereits, dass ein Bewusstsein für die Problematik vorhanden ist. Im weiteren Verlauf werden nun die unterschiedlichen Abschnitte konkret untersucht.

6.3.1. Kommunikation

Wenn es um die Verantwortung geht, die Kommunikation aufrechtzuerhalten, waren sich 75 % der Befragten darin einig, dass die Verantwortung beim gesamten Team liegt. Hierfür wurden überwiegend Meetings und persönliche Gespräche als Kommunikationskanäle genutzt. Die deutliche Mehrheit der Befragten empfand die regelmäßige und gemeinsame Kommunikation als einen großen Vorteil, so dass alle Beteiligten ihre Vorschläge und Ideen gut in das Projekt einbringen konnten. Außerdem waren sich alle einig, dass eine zu geringe Kommunikation Auswirkungen auf die Qualität des Produktes mit sich bringt. Was dabei besonders auffällt, ist, dass alle Personen mit dem Umfang und der Qualität der Kommunikation zufrieden waren, obwohl der Designexperte immer ein freiberuflich tätiger Designexperte war und somit stets räumlich getrennt vom Entwicklungsteam gearbeitet hat. Auf dieser Grundlage ist die Integration des agilen Designs mit seinen Artefakten und Etappen, was die Kommunikation anbelangt, als Erfolg zu werten.

6.3.2. Zusammenarbeit

Alle Befragten stimmen überein, dass die Verantwortung zur Aufrechterhaltung der Zusammenarbeit bei ihnen selbst liegt. Die Befragten waren mit der Geschwindigkeit und der Häufigkeit des Feedbacks des Product Owners sehr zufrieden. Dies setzt aber zwei wichtige

Faktoren voraus: Erstens muss der Product Owner die nötige Zeit haben und zweitens muss er regelmäßig an allen Meetings teilnehmen.
Trotz der räumlichen Trennung des Designexperten hat dieser laut allen Befragten zuverlässig und schnell auf Anfragen der Entwickler reagiert. Hieraus sieht man, dass nach einer bestimmten Zeit durch die immer wiederkehrenden Zyklen in der Kommunikation und die fest verankerte Rückkopplung der Informationen eine Art Routine in den Prozess integriert werden kann und der Designexperte dennoch seinen weiteren Projektverantwortungen nachgehen kann, ohne negative Auswirkungen auf den agilen Prozess zu haben. Hierfür ist laut allen Befragten eine regelmäßige Teilnahme aller Beteiligten an den für das Projekt vorgesehenen Meetings unabdingbar. Allgemein wurde die Zusammenarbeit als gut empfunden. Allerdings ist sie noch verbesserungswürdig, haben doch 30 % der Befragten sie nur als befriedigend bewertet.

6.3.3. Prozess

Aus der Befragung ging hervor, dass der Designexperte während des Projektes stets an allen Meetings wie Planungs-, Konzeptions-, Review- und retrospektiven Meetings beteiligt war. Ein Punkt jedoch sticht besonders hervor: Nur in einem Fall war der Designexperte am Stand-up-Meeting beteiligt. An dieser Stelle wurde Verbesserungspotential identifiziert. Bei einem solchen Meeting synchronisieren sich alle Projektbeteiligten auf den aktuellen Stand ihrer Arbeit, und es ist enorm wichtig, dass der Designexperte ebenfalls teilnimmt.
Alle Befragten waren sich darüber einig, dass die Time-Boxen der agilen Softwareentwicklung dazu beigetragen haben, die angestrebten Zwischenziele des Projektes innerhalb der dafür vorgesehenen Zeit fertigzustellen. Des Weiteren konnte laut allen Befragten die Qualität des Produktes während der gesamten Projektlaufzeit gesteigert werden. Das inkrementelle Design wurde unter anderem als dafür verantwortlich angesehen: Aufgrund seiner Existenz konnte man leichter auf Designänderungen reagieren und hatte somit mehr Zeit für die Implementierung. Somit lastete der Aufwand bei einem plötzlichen Kurswechsel nicht mehr nur auf dem Design an sich.
Ein weiterer positiver Aspekt fiel in der Geschwindigkeit des Teams auf. Im Durchschnitt betrug die Größe des Scrum-Teams vier Leute. Dabei hat sich die Geschwindigkeit jedoch nicht besonders auffällig verändert. Zu Anfang ist sie gestiegen, was normal ist. Im weiteren Verlauf des Projektes stieg sie kontinuierlich, aber nur leicht an. Am Ende ist sie dann wieder gefallen, was viele verschiedene Ursachen haben kann — Entwickler wurden abgezogen oder es wurden zum Ende hin noch einmal große Features ausgeliefert, die zeitintensiv waren. Zu beachten ist, dass das agile Design nicht die Geschwindigkeit des Projektes steigern soll, es

würde ansonsten schnell die Qualität des Produktes darunter leiden. Aufgrund der gleichbleibenden Geschwindigkeit lässt sich die Kontrolle über das Projekt beibehalten, was als ein weiterer positiver Aspekt bei der Integration der neuen Artefakte und Etappen zu werten ist.

6.3.4. Tools

Produktvision
Alle Beteiligten teilten die gleich Sicht und das gleiche Verständnis, was die Produktvision angeht. Die Befragten formulierten die Produktvision als ein kurz und schlüssig formuliertes Ziel mit Abgrenzungen zur Konkurrenz. Sie einigten sich darauf, dass die Produktvision eine gemeinsame Sicht auf das große Ganze liefern soll. Dabei hatten 63 % der Befragten bereits Erfahrungen mit dem Erstellen einer Produktvision sammeln können.
Bei fast allen beteiligten Personen wurde die Vision während der Projektlaufzeit reflektiert. Hierbei ist zu beachten, dass ein konsistentes Design und das dazugehörende Benutzerinterface direkt von der Produktvision betroffen sind — schließlich muss das Produkt nach außen hin das widerspiegeln, was es im Hintergrund macht, und zum Gesamtkonzept passen. Die überwiegende Mehrheit von 88 % ist sich darüber im Klaren, dass die Produktvision eine klare Vorstellung vom Endprodukt geschaffen hat. Dadurch ist es einfacher, nachhaltige Designentscheidungen zu treffen.

Styleguide
Der Styleguide wurde von den Befragten als Implementierungsartefakt bezeichnet, welches die wesentlichen Designelemente in Form eines lauffähigen Codes beinhaltet und stets aktuell ist und somit nicht separat von der Produktionsumgebung verwendet wird. Ein Drittel der Personen hatte noch keine Erfahrungen mit einem solchen Styleguide gemacht, dennoch funktionierte die Verwendung fast problemlos. Es hat sich gezeigt, dass der Styleguide am Anfang relativ intensiv genutzt, im weiteren Verlauf der Projektlaufzeit jedoch nur unregelmäßig gepflegt wurde.
Ganze 75 % identifizierten den Styleguide als Kommunikationsschnittstelle zwischen dem Designexperten und den Entwicklern. Darüber hinaus sah fast die Hälfte der Befragten eine Vereinfachung der Designimplementierung als Resultat der Verwendung eines solchen Styleguides. Unabhängig davon äußerten sich 82 % der Befragten zum technischen Know-how des Designexperten. Ihrer Aussage zufolge ist es zwingend notwendig, dass ein Designexperte über das nötige technische Wissen verfügt. Genau bei diesem Aspekt scheint es noch einen hohen Nachholbedarf zu geben, weil dieses geforderte Wissen noch nicht weit genug

verbreitet zu sein scheint. Ein Grund, warum der Styleguide im späteren Verlauf nicht mehr regelmäßig gepflegt wird, lässt sich in dem fehlenden technischen Wissen finden. Vielleicht ändert sich dies, sobald ein Designexperte mit technischem Wissen verfügbar ist und dadurch die Pflege zu seiner Aufgabe wird.

Paper Prototyping
Die Mehrheit der Teilnehmer hat bereits mit Papierprototypen gearbeitet, wobei laut den Angaben der Teilnehmer nur selten ein solches Verfahren verwendet wird. Im Umfeld der Befragten kamen die Papierprototypen meistens im Konzeptionsmeeting zum Einsatz. Dabei wurden aber nur sehr selten reelle Benutzertests durchgeführt, was oft am begrenzten Projektbudget lag. Sobald nun aber Papierprototypen in einem Projekt verwendet wurden, flossen zu 62 % die Resultate aus diesem Test direkt in den weiteren Verlauf der Implementierung ein. Die überwiegende Mehrheit von 76 % vertritt die Meinung, dass das Erstellen von Papierprototypen zur Verbesserung der Kommunikation beiträgt.

7. Zusammenfassung

In diesem Kapitel wird noch einmal das Problem zusammengefasst und die in der vorliegenden Arbeit entworfene Lösung vorgestellt und bewertet. Zudem soll ein Ausblick auf mögliche Weiterentwicklungen der hier aufgeführten Lösung vorgestellt werden.

7.1. Fazit

Scrum als agiles Projektmanagement-Framework sieht cross-funktionale Teams vor, um eine Aufgabe effizient bearbeiten und lösen zu können. In der Praxis jedoch zeigt sich häufig der Fall, dass externe Projektbeteiligte sich nur schwer in einen solchen Prozess integrieren lassen. Vorliegend wurde die Zusammenarbeit zwischen Entwicklern und Designexperten konkret untersucht, und in diesem Zusammenhang konnten mehrere Probleme im Projektverlauf identifiziert werden, die einen unterbrechungsfreien Prozess behindern. Es wurden Konfliktpotentiale offengelegt, die die Zusammenarbeit und die erfolgreiche Kommunikation zwischen beiden Rollen verhindern.

Durch eine tätigkeitsspezifische Verarbeitung und fehlende Kommunikation werden Teile des Projektes entwickelt, die sich später nur schwer umsetzten lassen oder erst gar keine Verwendung finden, weil sich in der Zwischenzeit Anforderungen geändert haben. Dieser Problematik liegt die parallele Verarbeitung der Aufgaben zugrunde: Der Designexperte ist am Anfang eines Projektes kurz präsent und zieht sich dann zurück um — vom Team isoliert — alleine ein Design für die Anwendung zu erstellen. In der Zwischenzeit ändern sich unter Umständen Anforderungen in der Anwendung oder bei der technischen Umsetzung, das nötige Feedback fließt jedoch nicht mehr in das Design ein, weil der Designexperte nicht Teil des Teams ist und somit der Informationsfluss unterbrochen wird. Oft ist dann die Zusammenführung beider Teile sehr zeit- und kostenintensiv, weil sich beide Teile aufgrund ihrer orthogonalen Verarbeitung unterschiedlich entwickelt haben.
Es wurde hier kein neuer Prozess erarbeitet, der diese Probleme beheben soll, es wurden lediglich neue Artefakte und Etappen im agilen Prozess etabliert. Für die im Prozess aufgedeckten Schwächen in der Kommunikation und der Zusammenarbeit eines

Designexperten mit dem Entwicklerteam wurden konkrete Lösungsansätze vorgeschlagen, die sich in der Praxis bewährt haben.

Ein gemeinsames Verständnis und eine Sicht auf das große Ganze der Anwendung bilden die Grundlage. Hierfür wird ein Sprint-Zero eingeführt. Dieses ermöglicht es dem Team, sich vor Beginn des Projektes besser kennenzulernen und in einem Elevator Pitch zusammen mit dem Kunden eine Projektvision zu erstellen. Die Praxis hat gezeigt, dass es bereits eine Verbesserung im Prozessfluss gibt, wenn Designexperten immer genau einen Sprint vor dem Entwicklerteam die gestalterischen Ausarbeitungen für den kommenden Sprint der Entwickler durchführen. Hierfür wurde ein wöchentliches Konzeptionsmeeting in den Prozess integriert. Dieses Meeting ermöglicht es den beteiligten Parteien, gemeinsam den kommenden Sprint vorzubereiten, indem der Designexperte zusammen mit den Kunden die zu implementierenden Oberflächen skizziert, während das Entwicklerteam hier als technischer Ansprechpartner dient und Feedback zur Machbarkeit gibt. Bei komplexeren Abläufen der Anwendungen werden Papierprototypen erstellt und gemeinsam mit allen Projektteilnehmern Usability Tests durchgeführt. Papierprototypen sind kostengünstig und können relativ schnell konkretes Feedback im Hinblick auf Machbarkeit und Umfang liefern. Außerdem sind sie ein hervorragendes Kommunikationswerkzeug, um komplexe Interaktionen im Design zu besprechen.

Im weiteren Verlauf werden nun sogenannte Pair Designing Sessions durchgeführt, bei denen Entwickler und Designexperten an einem Bildschirm sitzen und die Umsetzung des Designs gemeinsam durchführen. Die Grundlage hierfür bietet ein interaktiver Styleguide. Dieser Styleguide liefert ein Standardpaket an Komponenten, und der Designexperte und der Entwickler haben die Aufgabe, gemeinsam diesen Styleguide weiterzuentwickeln und die für die Anwendung nötigen Elemente zuerst im Styleguide zu erzeugen, um sie dann später in der Anwendung einsetzten zu können. Somit bietet der Styleguide eine gemeinsame Kommunikationsgrundlage, bei der der Designexperte auch ohne technisches Wissen mit dem Entwickler zusammenarbeiten kann.
Im weiteren Verlauf kann der Entwickler aber auch auf Elemente zurückgreifen, von denen er sicher ist, dass sie funktionieren und wie geplant eingesetzt werden können, weil er sich vorher mit dem Designexperten darauf geeinigt hat. Der Styleguide ist das Herzstück dieser Optimierung, er ermöglicht es, das Design inkrementell und zeitgleich zur Entwicklung zu erarbeiten. Somit schafft man von Anfang bis Ende sowohl Wert für den Kunden als auch für die Anwendung an sich, weil man immer genau so viel Vorarbeit leistet, wie benötigt wird. Überflüssige Arbeit, die zeitliche Ressourcen frisst und den Aufwand erhöht, ohne

tatsächlichen Wert für das Produkt zu erzeugen, wird dadurch verringert. Des Weiteren senkt der Styleguide die Parallelverarbeitung indem er alle Nachteile eines tätigkeitsspezifischen Sprint beseitigt.

Aufgrund der neuen Artefakte und Etappen wird der Designexperte stärker an das Team gebunden und als solches akzeptiert. Er kann sich somit auch besser mit seiner eigenen Arbeit identifizieren; es geht nicht mehr um einen Kampf, bei dem der Designexperte seine Entwürfe verteidigen muss, sondern um eine Zusammenarbeit, bei der das Design gemeinschaftlich und mit viel Kommunikation erarbeitet und stetig weiterentwickelt wird. Die Untersuchung verschiedener wissenschaftlicher Arbeiten hat gezeigt, dass im Ansatz der hier vorgestellte Weg bereits durchgeführt wird und in der Praxis bestehen kann. Durch die eigene langjährige Erfahrung bei der Entwicklung von Web-Anwendungen kann der Verfasser dem hier vorgestellten Vorgehen zustimmen. Bei seinem letzten Arbeitgeber, der mindmatters GmbH & Co. KG[1], wurde diese Vorgehensweise vor einem Jahr eingeführt und ließen sich damit die Qualität der Produkte, die Effektivität des Teams und die Zufriedenheit bei allen Projektteilnehmern deutlich verbessern.

7.2. Ausblick

Die vorliegende Arbeit beschäftigt sich mit der Integration eines Designexperten in ein agiles Umfeld. Hierfür wurde Scrum als agiles Vorgehensmodell herangezogen. Weiterführend wurde nur der Fall betrachtet, bei dem der Designexperte ein freiberuflich tätiger Designexperte ist und zu einem Projekt dazugekauft wurde; als externer Projektteilnehmer steht er dem Team nur unregelmäßig zur Verfügung. Es besteht nun die Möglichkeit, den Fall zu untersuchen, dass der Designexperte ein tatsächlich voll integriertes Teammitglied ist und den Entwicklern stets zur Verfügung steht. Dabei stellt sich die Frage, ob dann die hier vorgeschlagenen Etappen und Artefakte in ihrer Zusammenstellung funktionieren, wie man diese Artefakte und Etappen auf die neue Situation anpassen müsste oder ob diese neuen Methoden vielleicht unter diesen Umständen überflüssig sind und Scrum dann so funktioniert, wie es funktionieren soll. Ein weiteres Feld, welches untersucht werden könnte, ist der Bereich anderer Vorgehensmodelle, wie z. B. der Ansatz von Lean, auch unter Lean UX bekannt. Diese Methode beinhaltet im Ansatz die in der vorliegenden Arbeit genannten Teile, z. B. das Konzipieren und Erstellen von Prototypen, die dann von Benutzern getestet werden. Jedoch entstehen aufgrund der fehlenden Meetings und der fehlenden empirischen Prozesskontrolle neue Herausforderungen, die untersucht werden könnten.

[1] Software Dienstleistungsunternehmen aus Hamburg, url: http://www.mindmatters.de

Fragebogen zur Fallstudie über „Agile Design“

Nr.	Fallstudie: Agile Design - Allgemeines
1	Welche Rolle haben Sie beim aktuellen oder letzten agilen Projekt übernommen? Mögliche Antworten: Scrum Master, Product Owner, Konzeptioner, Entwickler, Designer, Weiß ich nicht
2	Wie hoch ist Ihre bisherige Beteiligung an Projekten? Mögliche Antworten: 1-2, 3-5, 6-8, mehr als 8, weiß ich nicht
3	Wie lange haben Sie schon Erfahrungen in agilen Prozessen? Mögliche Antworten: weniger als ein Jahr, 2-3 Jahre, 4-5 Jahre, mehr als 5 Jahre, weiß ich nicht
4	Nennen Sie 4 Werte, die Ihrer Meinung nach Agilität ausmachen! Mögliche Antworten: nur 4 Schlagwörter die Ihnen einfallen aufschreiben
5	Haben Sie vor Ihrem letzten Projekt schonmal vom agilen Design gehört? Mögliche Antworten: Ja, Nein
6	Wie hoch schätzen Sie die Risiken, bezogen auf den Projekterfolg, in der Zusammenarbeit mit Projektteilnehmern ein, die nicht ins Scrum Team integriert sind? Mögliche Antworten: Hohes Risiko, teilweises Risiko, eher geringe Risiken, keine Risiken, weiß ich nicht
7	Wie wirkt sich Ihrer Meinung nach der Einsatz eines Teammitglied der nicht ins Scrum Team integriert ist, auf den agilen Prozess aus? Mögliche Antworten: Beantwortung in ein oder zwei Sätzen oder durch Angabe bestimmter Schlagwörter

Nr.	Fallstudie: Agile Design - Kommunikation
1	Wer ist für das aufrechterhalten der Kommunikation während eines agilen Projektes verantwortlich? *Mögliche Antworten: Product Owner, Scrum Master, Scrum Team, Alle, Weiß ich nicht*
2	Welche Tools zur Kommunikation wurden im aktuellen oder letzten agilen Projekt eingesetzt? Mögliche Antworten: Mail, Telefon, Instant Messenger, Meetings, Andere
3	Wie gut konnten alle beteiligten ihre Ideen in das aktuelle oder letzte Projekt einbringen? Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht
4	“Während des Sprints wurde häufig und regelmäßig zwischen den Projektbeteiligten kommuniziert.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
5	“Der Product Owner war immer erreichbar wenn man ihn gebraucht hat.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
6	“Fehlende Kommunikation im Projekt trägt zu einer schlechteren Qualität des Produktes bei.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
7	War der Designer räumlich vom Entwickler-Team getrennt? Mögliche Antworten: Ja, Nein
8	Bewerten Sie die Kommunikation in vorherigen Projekte! Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht
9	Bewerten Sie die Kommunikation beim letzten Projekt! Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht

Nr.	Fallstudie: Agile Design - Zusammenarbeit
1	Wer ist für das aufrechterhalten der Zusammenarbeit während eines agilen Projektes verantwortlich? Mögliche Antworten: Product Owner, Scrum Master, Scrum Team, Alle, Weiß ich nicht
2	Wie war die Geschwindigkeit des Feedbacks des Product Owners? Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht
3	Wie häufig gab der Product Owner sein Feedback ab? Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht
4	War der Designer ein Freelancer? Mögliche Antworten: Ja, Nein
5	Hat der Designer schnell und zuverlässig auf Anfragen reagiert? Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
6	Welche Rollen waren an der Konzipierung und Erstellung von Mockups der einzelnen Features beteiligt? Mögliche Antworten: Product Owner, Entwickler, Designer, Konzeptioner, Weiß ich nicht
7	Wie war die Zusammenarbeit zwischen Design und Entwicklung? Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht
8	Wie war die Zusammenarbeit der einzelnen Teammitglieder insgesamt? Mögliche Antworten: Sehr gut, Gut, Befriedigend, Mangelhaft, Weiß ich nicht

Nr.	Fallstudie: Agile Design - Prozess
1	An welchen Meetings war der Designer beteiligt? Mögliche Antworten: Stand-Up, Planung, Backlog grooming, Konzeption, Review, Retrospektive
2	“Der Product Owner war voll in den Prozess integriert.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
3	“Der agile Prozess hat dazu beigetragen bzw. geholfen, Termine einzuhalten.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
4	“Die Qualität des Produktes konnte (kontinuierlich) gesteigert werden.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
5	“Es war einfach, auf Design-Änderungen zwischen den Sprints zu reagieren.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
6	Wie würden Sie die Anzahl der Meetings bewerten? Mögliche Antworten: Zu viel, Genau richtig, Eher zu wenig, Zu wenig, Weiß ich nicht
7	Wenn Sie ein Meeting abschaffen könnten, welches wäre das? Mögliche Antworten: Stand-Up, Planung, Backlog grooming, Konzeption, Review, Retrospektive, Keins
8	Wie wurde mit neuen Features oder der Neupriorisierung von bestehenden Features im Prozess umgegangen? Beantwortung in ein oder zwei Sätzen oder durch Angabe bestimmter Schlagwörter
9	Fand eine iterative Weiterentwicklung des Designs statt? Mögliche Antworten: Ja, Nein
10	Wie groß war das Scrum-Team? Anzahl der Teammitglieder am Anfang, Minimum und Maximum
11	Wie hat sich die velocity des Teams während der Projektlaufzeit entwickelt? Am Anfang, in der Mitte und am Ende des Projektes (gesteigert, gleich geblieben, gesunken, keine Ahnung)

Nr.	Fallstudie: Agile Design - Tools (Produktvision)
1	Was verstehen Sie unter einer Produktvision? Möglichst kurzfassen
2	Haben Sie schon einmal in einem Elevator Pitch eine Produktvision erstellt? Mögliche Antworten: Ja, Nein
3	Wurde die Vision während der Projektlaufzeit reflektiert? Mögliche Antworten: Ja, Nein
4	Wurde die Vision während der Projektlaufzeit angepasst? Mögliche Antworten: Ja, Nein
5	“Die Produktvision hat am Beginn des Projektes dazu beigetragen, eine klarere Vorstellung vom geplanten Endprodukt zu haben.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
6	“Die Produktvision war deutlich und verständlich formuliert.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
7	“Die Produktvision war kurz und bündig formuliert.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
8	“Die Produktvision hat dazu beigetragen, dass Entscheidungen leichter und effektiver gefällt werden konnten.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht

Nr.	Fallstudie: Agile Design - Tools (Styleguide)
1	Was verstehen Sie unter einem interaktiven Styleguide? Möglichst kurzfassen
2	Haben Sie bereits in anderen Projekten mit einer interaktiven HTML Bibliothek zu tun? Mögliche Antworten: Ja, Nein
3	“Der Styleguide wurde regelmässig refactored.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
4	“Der Styleguide wurde regelmäßig auf Vollständigkeit überprüft und gepflegt.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
5	“Der Styleguide eignet sich als technologische Schnittstelle zur Kommunikation zwischen Design und Technik.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
6	“Die Hürde zwischen Entwickler und Designer, also zwischen interaktivem HTML und statischen Grafik-Dateien konnte durch den Styleguide gesenkt werden.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
7	“Der Designer muss in einem agilen Projekt technisches Know-how, z.B. HTML- und CSS-Kenntnisse haben.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
8	Ist solches technisches Wissen der Designer ihrer Erfahrung nach vorhanden? Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht

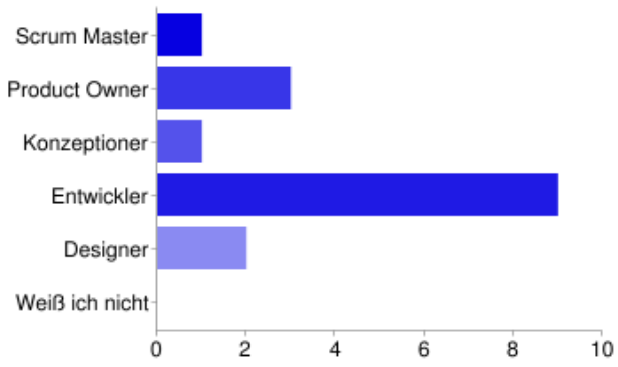
Nr.	Fallstudie: Agile Design - Tools (Paper Prototyping)
1	Was verstehen Sie unter Papierprototypen? Möglichst kurzfassen
2	Haben Sie schon einmal in Ihren Projekten Papierprototypen angewandt? Mögliche Antworten: Ja, Nein
3	Wie oft wurde Paper Prototyping im Projekt angewandt? Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht
4	Wurden während der Konzeptionsmeetings Papierprototypen erstellt? Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht
5	Kamen die erstellten Papierprototypen auch in Benutzertests zum Einsatz? Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht
6	Falls ja: Wurde das Feedback aus den Benutzertests in der Entwicklung und dem weiteren Projektverlauf berücksichtigt? Mögliche Antworten: Sehr oft, Oft, Selten, Kaum, Weiß ich nicht
7	“Das Erstellen von Papierprototypen hat zur Verbesserung der Kommunikation beigetragen.” Mögliche Antworten: Stimme voll zu, Stimme eher zu, Eher nicht, Gar nicht, Weiß ich nicht
8	Welche Vorteile ergeben sich ihrer Meinung nach durch das Erstellen von Papierprototypen? Nennen von 3 Stichwörter
9	Welche Nachteile ergeben sich ihrer Meinung nach durch das Erstellen von Papierprototypen? Nennen von 3 Stichwörter

Datenschutzerklärung

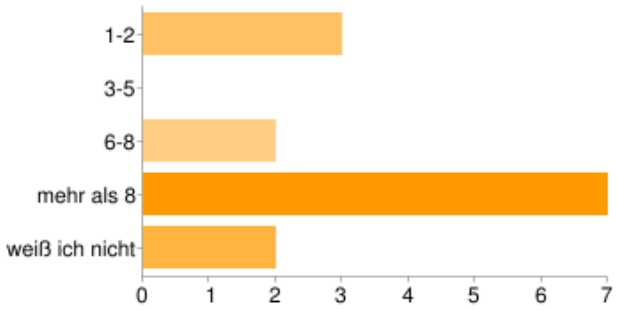
Die Befragung wird nach den gesetzlichen Bestimmungen des Datenschutzes in Deutschland durchgeführt. Die Teilnahme an der Befragung ist freiwillig. Die Daten werden nur in anonymisierter Form ausgewertet. Die Daten und Ihre Adresse werden getrennt und besonders gesichert gelagert. Die Ergebnisse der wissenschaftlichen Auswertung werden ausschließlich in anonymisierter Form und für Gruppen zusammengefasst dargestellt. Niemand kann also aus den Ergebnissen erkennen, von welcher Person die Angaben stammen.

Auswertung der Fallstudie über „Agile Design“

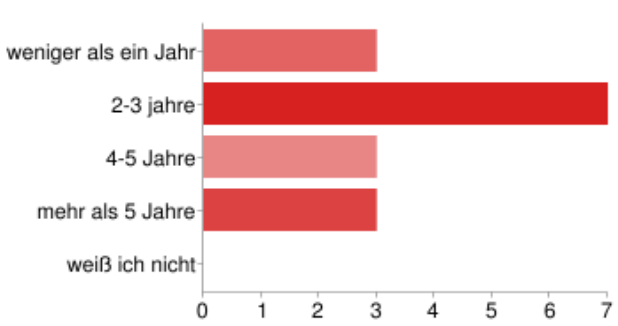
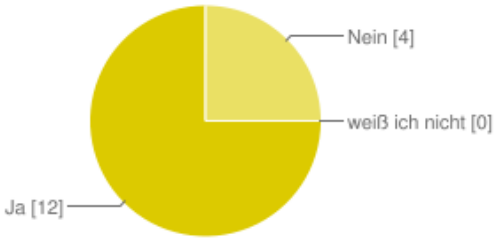
Nr.	Fallstudie: Agile Design - Allgemeines
1	Welche Rolle haben Sie beim aktuellen oder letzten agilen Projekt übernommen?
2	Wie hoch ist Ihre bisherige Beteiligung an Projekten?

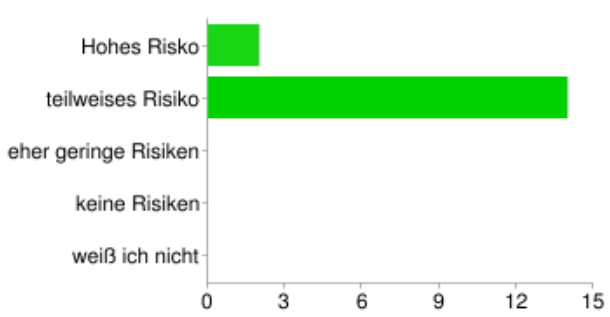


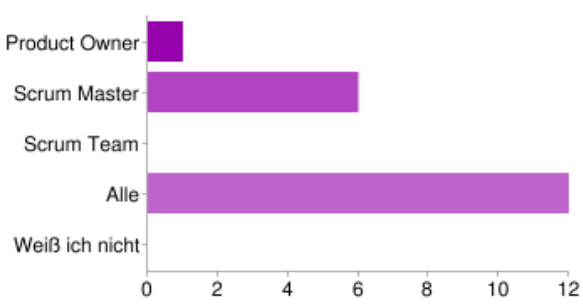
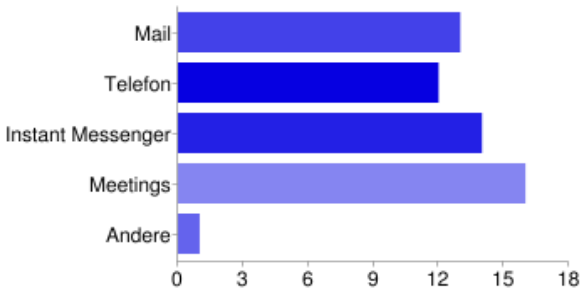
Scrum Master	1	6%
Product Owner	3	19%
Konzeptioner	1	6%
Entwickler	9	56%
Designer	2	13%
Weiß ich nicht	0	0%

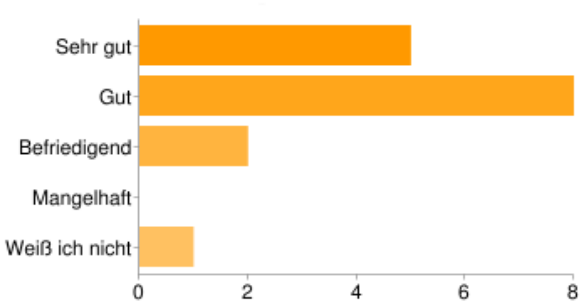
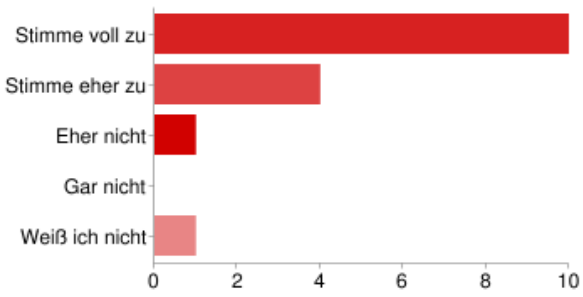
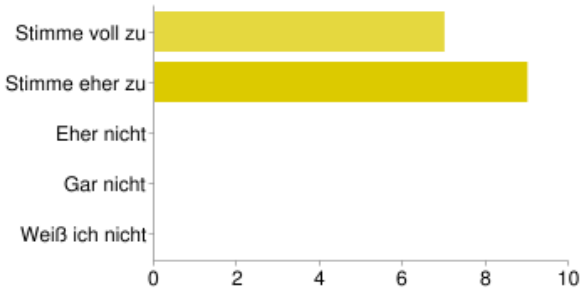


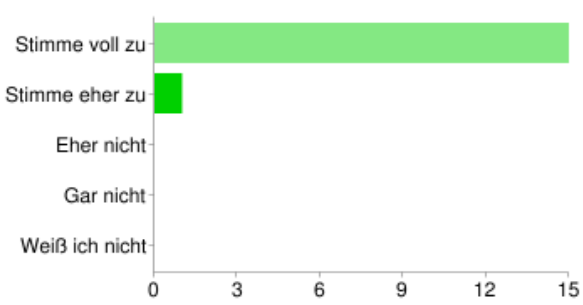
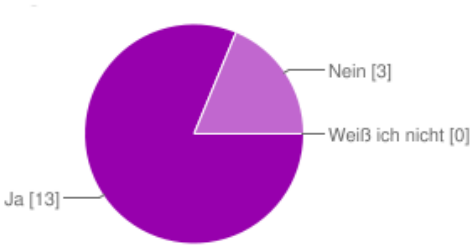
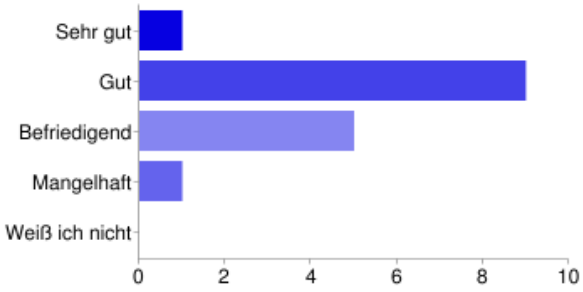
1-2	3	19%
3-5	0	0%
6-8	2	13%
mehr als 8	7	44%
weiß ich nicht	2	13%

Nr.	Fallstudie: Agile Design - Allgemeines
3	Wie lange haben Sie schon Erfahrungen in agilen Prozessen?  weniger als ein Jahr 3 19% 2-3 jahre 7 44% 4-5 Jahre 3 19% mehr als 5 Jahre 3 19% weiß ich nicht 0 0%
4	Nennen Sie 4 Werte, die Ihrer Meinung nach Agilität ausmachen! • Kundennah, kurze Zyklen, Selbstständigkeit, Flexibilität • Feedback, Kommunikation, Einfachheit, Mut • Flexibilität, Transparenz, Planungssicherheit, Dynamik • transparenz, direkte kommunikation, feedbackloop, moderation • Kommunikation, Neuentwicklung, Teamwork, Flexibilität • Mut, Einfachheit, Feedback, Kommunikation • Flexibilität, Pair Programming, Selbstständiges Arbeiten, Unbürokratisch • Kommunikation, Mut, Disziplin, Flexibilität • Flexibilität, Qualität, Schneller Einsatz der Software, Transparenz • Einfachheit, Kommunikation, Mut, Feeback • Einfachheit, Kommunikation, Feedback und Mut • Schnelligkeit, Balance von Budget, Anforderungen, Zeit, regelmäßige Kommunikation, Coaching des Teams • Systemdenken, Kommunikation, Feedback, Lernen • Feedback, Kommunikation, Einfachheit, Mut • Kundennah, kurze Zyklen, Selbstständigkeit, Flexibilität • Transparenz, Mut, Kommunikation, Feedback
5	Haben Sie vor Ihrem letzten Projekt schonmal vom agilen Design gehört?  Ja 12 75% Nein 4 25% weiß ich nicht 0 0%

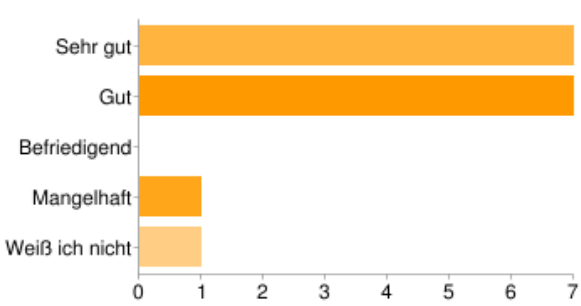
Nr.	Fallstudie: Agile Design - Allgemeines
6	Wie hoch schätzen Sie die Risiken, bezogen auf den Projekterfolg, in der Zusammenarbeit mit Projektteilnehmern ein, die nicht ins Scrum Team integriert sind?  Hohes Risiko 2 13% teilweises Risiko 14 88% eher geringe Risiken 0 0% keine Risiken 0 0% weiß ich nicht 0 0%
7	Wie wirkt sich Ihrer Meinung nach der Einsatz eines Teammitglied der nicht ins Scrum Team integriert ist, auf den agilen Prozess aus? • Erschwerte Kommunikation, sehr wichtig bei agilem Prozess. • potentieller Kultur(Werte)-Clash, mehr Koordination notwendig, Story-Umsetzungzeit erhöht sich, potentiell separate Stories für internes vs externes Team, spezielles Know-How durch externes Team-Mitglied • Enorm. Durch Nicht-Integration von nur einem einzigen Teammitglied verringern sich insbesondere Transparenz und Planungssicherheit. Dennoch ist "Bilderbuch-Scrum" in der Realität nicht immer umsetzbar. Ein klares Methodenset, wie mit möglichen Abweichungen umzugehen ist, wäre hilfreich. • großer overhead bei der implementierung • Wer nicht weiß was ansteht, weiß auch nicht was er tut und muss gegebenenfals alles noch einmal machen. Das nervt und frustriert. > Kommunikation! • mehr Abstimmungsoverhead, weniger inhaltliche Kohäsion, keine geteilte Sicht • Schlecht durch nicht optimale Kommunikation. • Inspirierend, hinderlich, in jeden Fall aufwändig • Der Wissensstand über aktuelle Tätigkeiten, Prozesse und Commitments ist unterschiedlich und führt zu Verzögerungen in der Umsetzung und führt nicht zu einer gleichen Sicht auf die Ziele und ggf. auf die Vision. • Als „Chicken“ sorgen solche Teammitglieder typischerweise gerne für (unnötigen) Wirbel, da sie nicht wirklich Verantwortung dafür übernehmen müssen (macht ja das Kernteam). Nicht direkt am Erfolg eines Projektes beteiligte „Entscheider“ stören und behindern die wahren Macher bei der Durchführung des Projekts. • Es kommt darauf an: Wenn es zu allen nötigen Meetings kommt und beim Daily Stand Up teilnimmt, dann könnte es klappen. Auf alle Fälle muss die Möglichkeit bestehen, dass das externe Projektmitglied und das Scrum Team auch außerhalb der Meetings kommunizieren • Der Prozess könnte verlangsamt werden oder fehlende Informationen könnten zu auseinander laufenden Zielbildern führen. • Relativ hoher Kommunikationsoverhead, Steigerung des Projektrisikos • potentieller Kultur(Werte)-Clash, mehr Koordination notwendig, Story-Umsetzungzeit erhöht sich, potentiell separate Stories für internes vs externes Team, spezielles Know-How • Erschwerte Kommunikation, sehr wichtig bei agilem Prozess. • Verzögerungen im Feedback, längere Projektlaufzeit, geringe kommunikation

Nr.	Fallstudie: Agile Design - Kommunikation
1	Wer ist für das aufrechterhalten der Kommunikation während eines agilen Projektes verantwortlich?  Product Owner 1 6% Scrum Master 6 38% Scrum Team 0 0% Alle 12 75% Weiß ich nicht 0 0% People may select more than one checkbox, so percentages may add up to more than 100%.
2	Welche Tools zur Kommunikation wurden im aktuellen oder letzten agilen Projekt eingesetzt?  Mail 13 81% Telefon 12 75% Instant Messenger 14 88% Meetings 16 100% Andere 1 6% People may select more than one checkbox, so percentages may add up to more than 100%.

Nr.	Fallstudie: Agile Design - Kommunikation
3	Wie gut konnten alle beteiligten ihre Ideen in das aktuelle oder letzte Projekt einbringen?  Sehr gut 5 31% Gut 8 50% Befriedigend 2 13% Mangelhaft 0 0% Weiß ich nicht 1 6%
4	“Während des Sprints wurde häufig und regelmäßig zwischen den Projektbeteiligten kommuniziert.”  Stimme voll zu 10 63% Stimme eher zu 4 25% Eher nicht 1 6% Gar nicht 0 0% Weiß ich nicht 1 6%
5	“Der Product Owner war immer erreichbar wenn man ihn gebraucht hat.”  Stimme voll zu 7 44% Stimme eher zu 9 56% Eher nicht 0 0% Gar nicht 0 0% Weiß ich nicht 0 0%

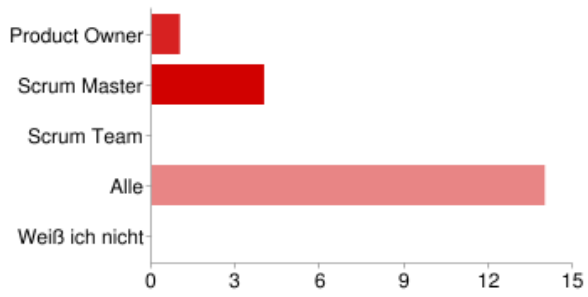
Nr.	Fallstudie: Agile Design - Kommunikation
6	“Fehlende Kommunikation im Projekt trägt zu einer schlechteren Qualität des Produktes bei.”  Stimme voll zu **15** 94% Stimme eher zu **1** 6% Eher nicht **0** 0% Gar nicht **0** 0% Weiß ich nicht **0** 0%
7	War der Designer räumlich vom Entwickler-Team getrennt?  Ja **13** 81% Nein **3** 19% Weiß ich nicht **0** 0%
8	Bewerten Sie die Kommunikation in vorherigen Projekte!  Sehr gut **1** 6% Gut **9** 56% Befriedigend **5** 31% Mangelhaft **1** 6% Weiß ich nicht **0** 0%

Nr.	Fallstudie: Agile Design - Kommunikation
9	Bewerten Sie die Kommunikation beim letzten Projekt!



Sehr gut	7	44%
Gut	7	44%
Befriedigend	0	0%
Mangelhaft	1	6%
Weiß ich nicht	1	6%

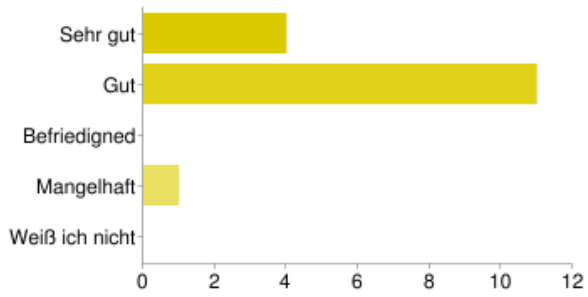
Nr.	Fallstudie: Agile Design - Zusammenarbeit
1	Wer ist für das aufrechterhalten der Zusammenarbeit während eines agilen Projektes verantwortlich?



Product Owner	1	6%
Scrum Master	4	25%
Scrum Team	0	0%
Alle	14	88%
Weiß ich nicht	0	0%

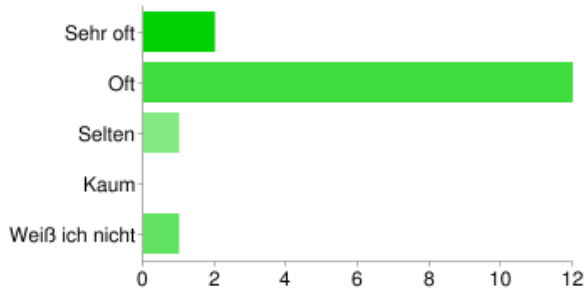
People may select more than one checkbox, so percentages may add up to more than 100%.

2	Wie war die Geschwindigkeit des Feedbacks des Product Owners?

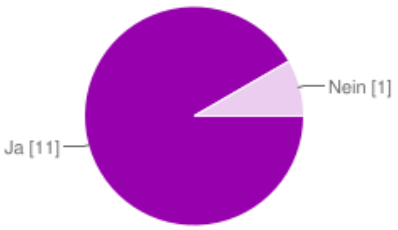
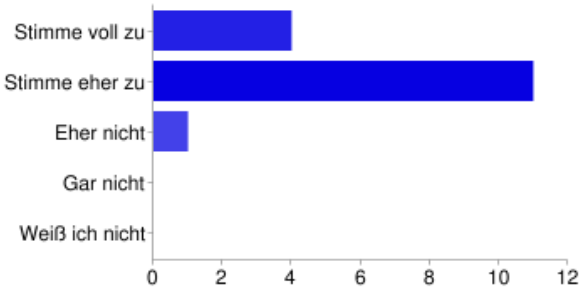
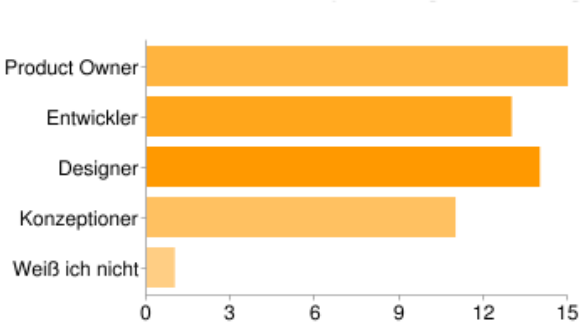


Sehr gut	4	25%
Gut	11	69%
Befriedigned	0	0%
Mangelhaft	1	6%
Weiß ich nicht	0	0%

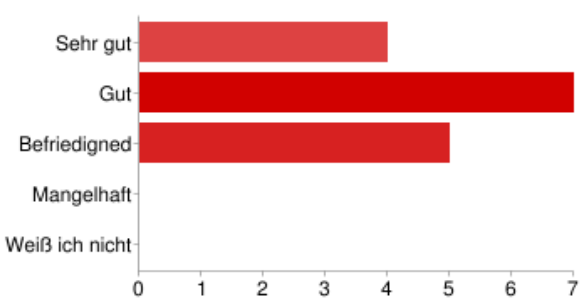
3	Wie häufig gab der Product Owner sein Feedback ab?



Sehr oft	2	13%
Oft	12	75%
Selten	1	6%
Kaum	0	0%
Weiß ich nicht	1	6%

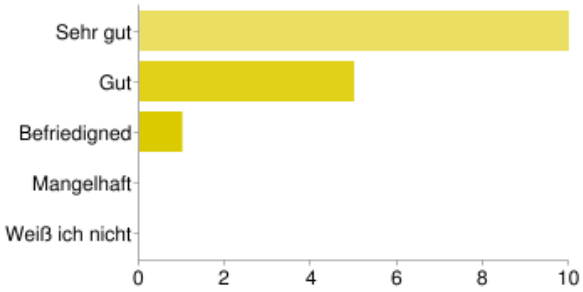
Nr.	Fallstudie: Agile Design - Zusammenarbeit
4	War der Designer ein Freelancer?  Ja 11 69% Nein 1 6%
5	Hat der Designer schnell und zuverlässig auf Anfragen reagiert?  Stimme voll zu 4 25% Stimme eher zu 11 69% Eher nicht 1 6% Gar nicht 0 0% Weiß ich nicht 0 0%
6	Welche Rollen waren an der Konzipierung und Erstellung von Mockups der einzelnen Features beteiligt?  Product Owner 15 94% Entwickler 13 81% Designer 14 88% Konzeptioner 11 69% Weiß ich nicht 1 6% People may select more than one checkbox, so percentages may add up to more than 100%.

Nr.	Fallstudie: Agile Design - Zusammenarbeit
7	Wie war die Zusammenarbeit zwischen Design und Entwicklung?

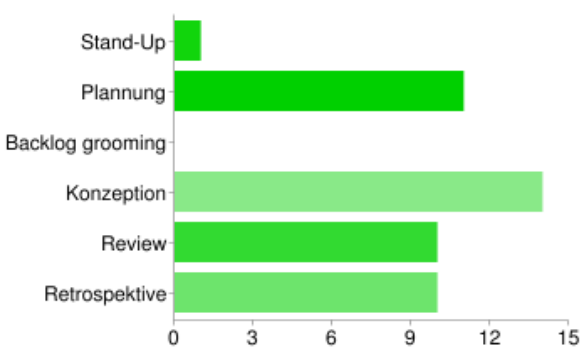
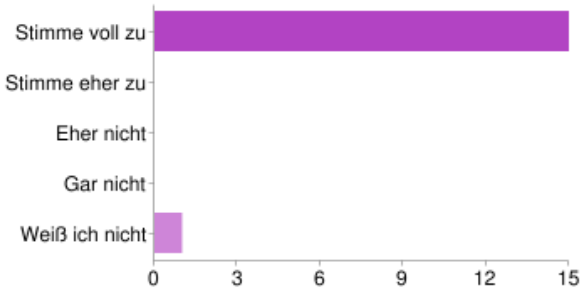
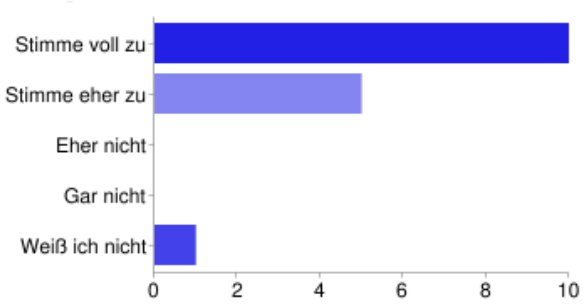


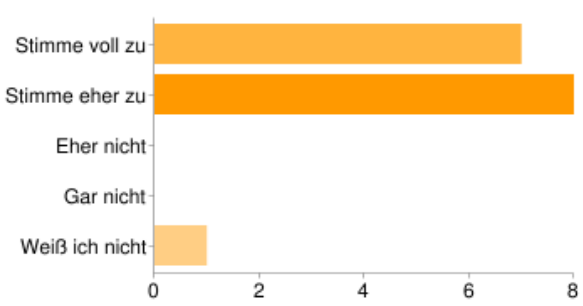
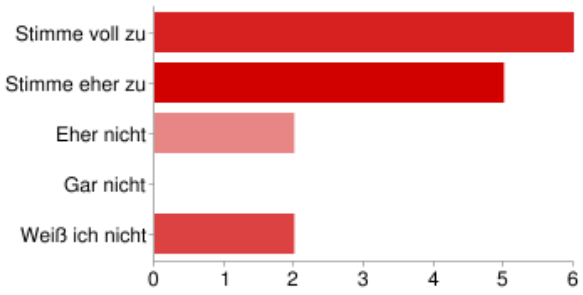
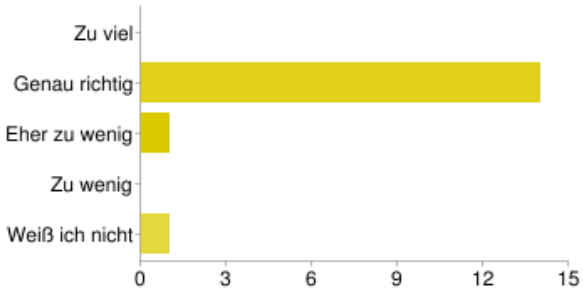
Sehr gut	**4**	25%
Gut	**7**	44%
Befriedigned	**5**	31%
Mangelhaft	**0**	0%
Weiß ich nicht	**0**	0%

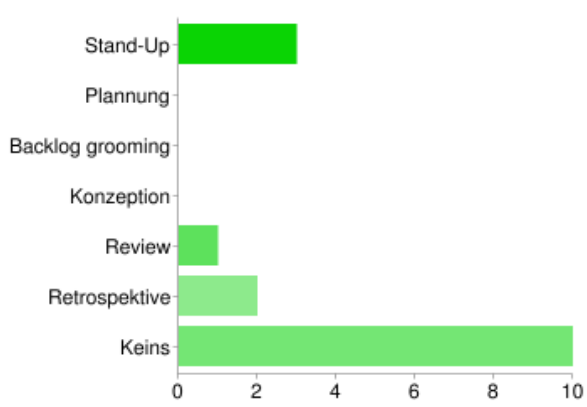
8	Wie war die Zusammenarbeit der einzelnen Teammitglieder insgesamt?



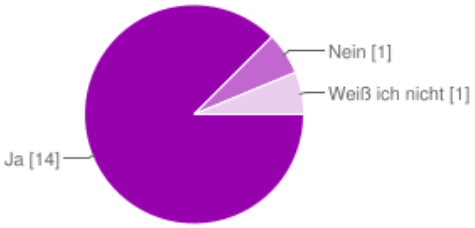
Sehr gut	**10**	63%
Gut	**5**	31%
Befriedigned	**1**	6%
Mangelhaft	**0**	0%
Weiß ich nicht	**0**	0%

Nr.	Fallstudie: Agile Design - Prozess
1	An welchen Meetings war der Designer beteiligt?  Stand-Up 1 7% Plannung 11 73% Backlog grooming 0 0% Konzeption 14 93% Review 10 67% Retrospektive 10 67% People may select more than one checkbox, so percentages may add up to more than 100%.
2	“Der Product Owner war voll in den Prozess integriert.”  Stimme voll zu 15 94% Stimme eher zu 0 0% Eher nicht 0 0% Gar nicht 0 0% Weiß ich nicht 1 6%
3	“Der agile Prozess hat dazu beigetragen bzw. geholfen, Termine einzuhalten.”  Stimme voll zu 10 63% Stimme eher zu 5 31% Eher nicht 0 0% Gar nicht 0 0% Weiß ich nicht 1 6%

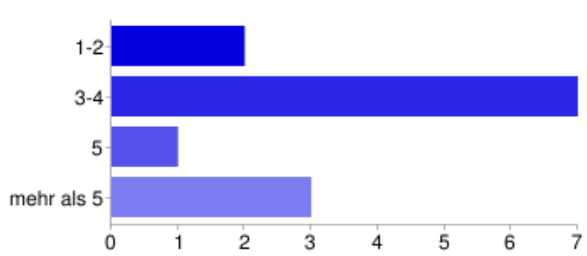
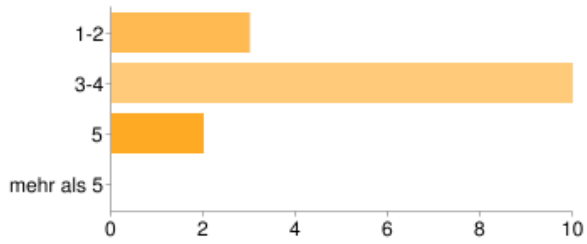
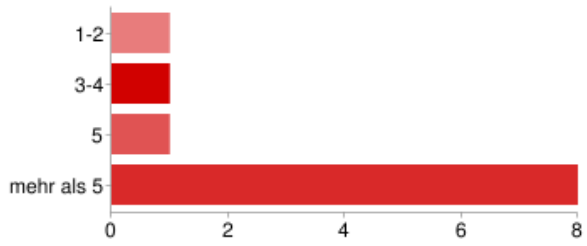
Nr.	Fallstudie: Agile Design - Prozess
4	“Die Qualität des Produktes konnte (kontinuierlich) gesteigert werden.”  Stimme voll zu 7 44% Stimme eher zu 8 50% Eher nicht 0 0% Gar nicht 0 0% Weiß ich nicht 1 6%
5	“Es war einfach, auf Design-Änderungen zwischen den Sprints zu reagieren.”  Stimme voll zu 6 38% Stimme eher zu 5 31% Eher nicht 2 13% Gar nicht 0 0% Weiß ich nicht 2 13%
6	Wie würden Sie die Anzahl der Meetings bewerten?  Zu viel 0 0% Genau richtig 14 88% Eher zu wenig 1 6% Zu wenig 0 0% Weiß ich nicht 1 6%

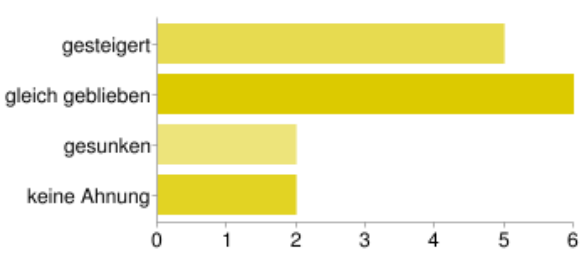
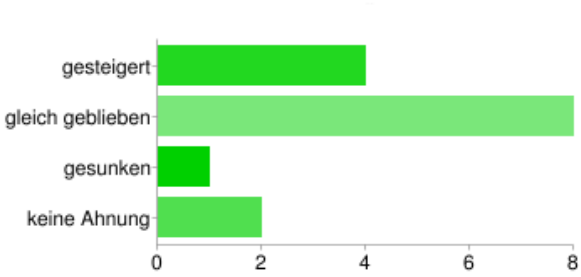
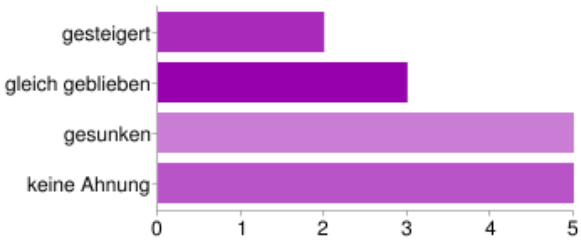
Nr.	Fallstudie: Agile Design - Prozess
7	Wenn Sie ein Meeting abschaffen könnten, welches wäre das?  Stand-Up 3 19% Plannung 0 0% Backlog grooming 0 0% Konzeption 0 0% Review 1 6% Retrospektive 2 13% Keins 10 63%
8	Wie wurde mit neuen Features oder der Neupriorisierung von bestehenden Features im Prozess umgegangen? • Kurzfristig Meeting anberaumt, neue Features/Priorisierung, fertig • Release-Planung, Sprint-Planung, Epics, Themenbezogene Sprints, Features zunächst in die Breite vs Tiefe realisiert • Neue Features oder Neupriorisierung wurde im Team diskutiert und entsprechend des Diskussionsergebnis umgesetzt" • Priorisierung, Konzeption -> Scribbles und Story-Ausarbeitung, Schätzen und Design-Ausarbeitung, Repriorisierung, Implementierung • Wurden einfach und problemlos integriert. • Eingabe ins Projektmanagement Tool, Kommunikation in den Meetings • Neue Features wurden geschätzt, priorisiert und im Sprint abgearbeitet. • "Neue Features: Erst konzeptionell entwickelt, dann geschätzt, evtl. danach neu konzeptioniert, nochmal geschätzt und dann vom PO einsortiert. • Bestehende Features: Im Vorfeld des Planning vom PO in die richtige Reihenfolge gebracht und das dann beim Planning erläutert." • Während des Sprints gab es keine Änderungen. Ansonsten konnten die soviele Features hinzugefügt werden, wie nötig. Auch die Priorisierung im Backlog konnte jederzeit geändert werden. • Neue Features sowie auch Neupriorisierungen wurden im Planungsmeeting mit ihren Konsequenzen auf das Gesamtprojekt besprochen. Neue Features wurden i.d.R. vorher grob beschrieben (Userstories). • Release-Planung, Sprint-Planung, Epics, Themenbezogene Sprints, Features zunächst in die Breite vs Tiefe realisiert • Kurzfristig Meeting anberaumt, neue Features/Priorisierung, fertig • wurden immer nur zwischen zwei sprints in das backlog gelegt und entsprechende priorisiert, neuprio. fand nie in einem sprint statt

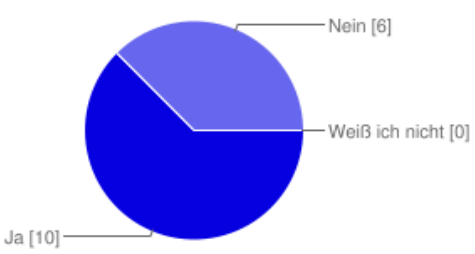
Nr.	Fallstudie: Agile Design - Prozess
9	Fand eine iterative Weiterentwicklung des Designs statt?

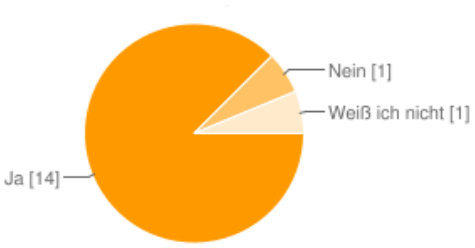
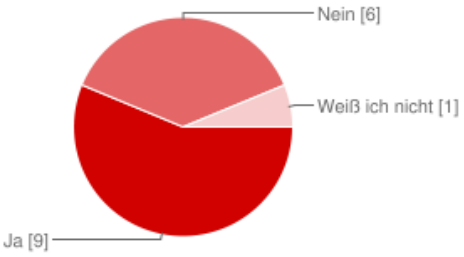
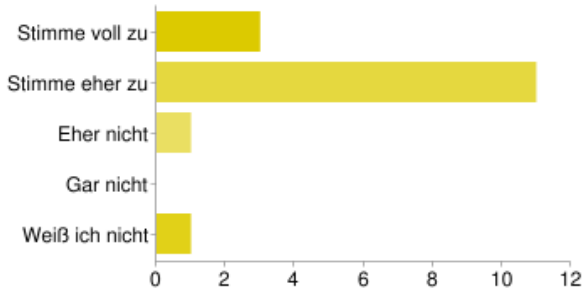


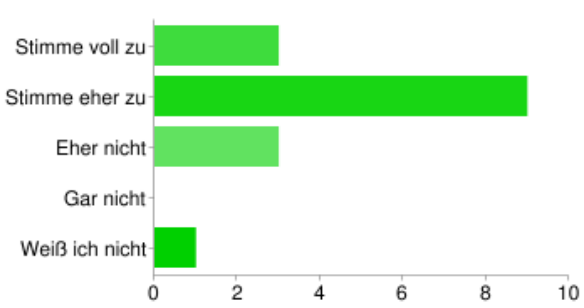
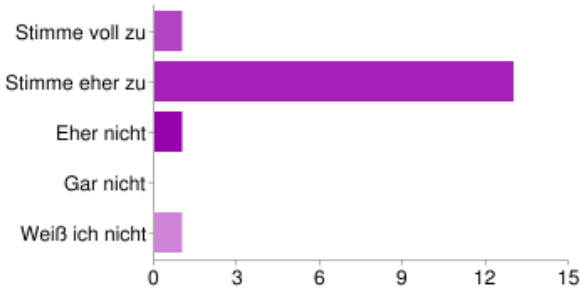
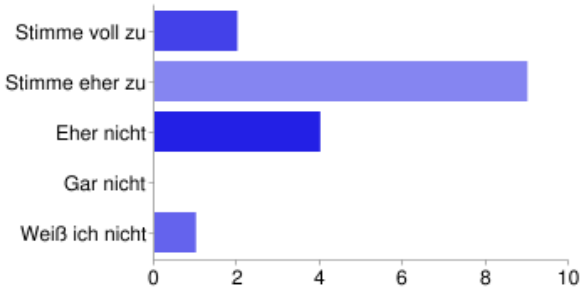
Ja	**14**	88%
Nein	**1**	6%
Weiß ich nicht	**1**	6%

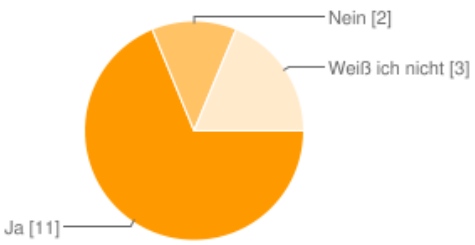
Nr.	Fallstudie: Agile Design - Prozess
10	Wie groß war das Scrum-Team? *Am Anfang des Projektes*  1-2 **2** 13% 3-4 **7** 44% 5 **1** 6% mehr als 5 **3** 19% *Minimum*  1-2 **3** 19% 3-4 **10** 63% 5 **2** 13% mehr als 5 **0** 0% *Maximum*  1-2 **1** 6% 3-4 **1** 6% 5 **1** 6% mehr als 5 **8** 50%

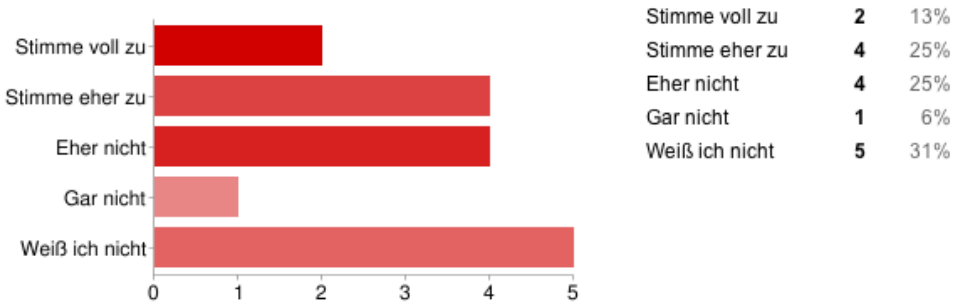
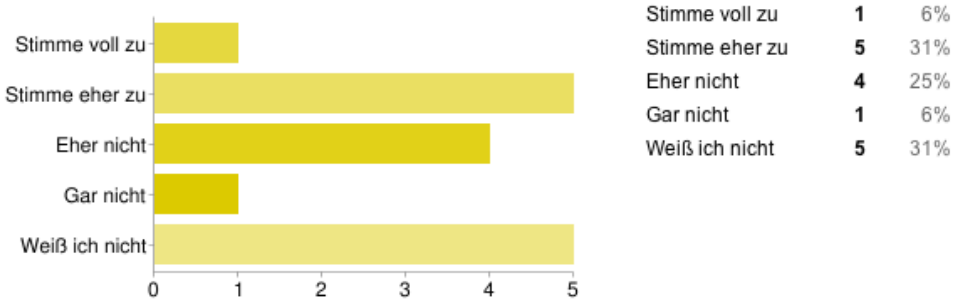
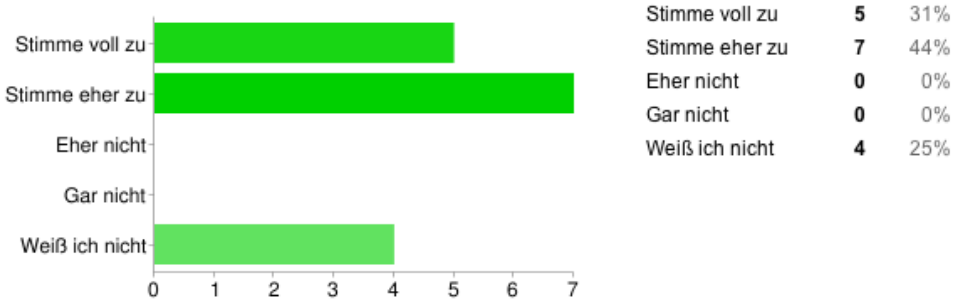
Nr.	Fallstudie: Agile Design - Prozess
11	Wie hat sich die velocity des Teams während der Projektlaufzeit entwickelt? *Anfang des Projektes*  gesteigert 5 31% gleich geblieben 6 38% gesunken 2 13% keine Ahnung 2 13% *In der Mitte des Projektes*  gesteigert 4 25% gleich geblieben 8 50% gesunken 1 6% keine Ahnung 2 13% *Am Ende des Projektes*  gesteigert 2 13% gleich geblieben 3 19% gesunken 5 31% keine Ahnung 5 31%

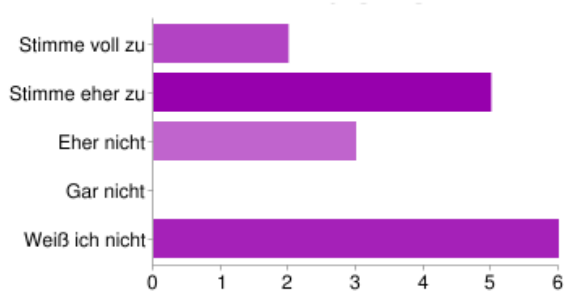
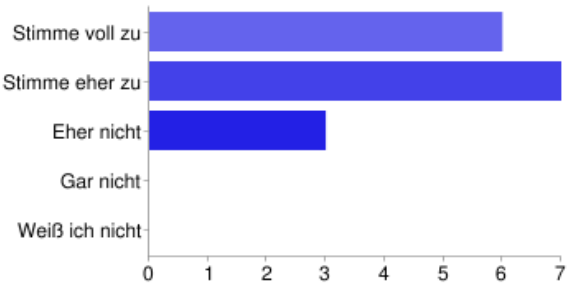
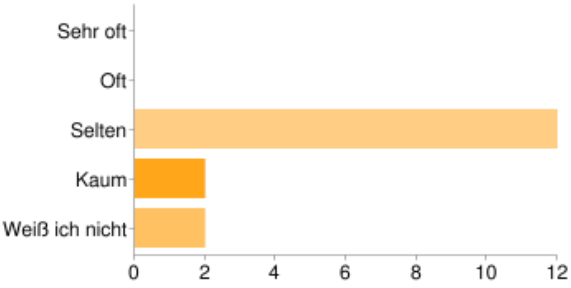
Nr.	Fallstudie: Agile Design - Tools (Produktvision)
1	Was verstehen Sie unter einer Produktvision? • Produkt in einem Satz beschreiben. • übergeordnete Ziel, identifiziert Bedürfnisse von Zielgruppen, nennt die Kernfeatures, Abgrenzung zur Konkurrenz • Übergreifende Produktbeschreibung inkl. Problemlösung, Zielsetzung, Zielgruppen und Geschäftsmodell, die allen Projektbeteiligten ein klares Produkt- und Zielbild gibt. • definition von use cases, user benefits und business model. • Wohin soll die Reise gehen. Das Ziel. • Ein geteiltes Produktbild, bestehend aus Elevator Pitch, Zielgruppendefinition inkl. Personae, Business Objectives und Epics. • Produkt in einem Satz beschreiben. • Das visuelle und funktionsoriente Gesamtbild einer Anwendung, inklusive Zielgruppenausrichtung. • Hier wird formuliert was ich machen möchte, welche Bedürfnisse abgedeckt werden, welche Kunden ich ansprechen und wie mein Geschäftsmodel aussieht. • Ein Leitfaden, der allgemein aber auch präzise genug formuliert ist, dass sich alle zukünftigen Features daran messen lassen, ob sie diesem Leitfaden genügen. Er dient als Entscheidungshilfe und muss in regelmäßigen Abständen aktualisiert und neubewertet werden. • Elevator Pitch, ausführlich Definition von Zielgruppe, Kernbedürfnisse, Features die diese befriedigen sollen, Konkurenzprodukte, USP, Monetarisierung • Das Zielbild eines zukünftigen Produktes, das Kunden aus bestimmten Gründen unbedingt haben wollen bzw. das eine Lücke schliesst. • Die Produktvision ist eine gemeinsame Sicht auf das Produkt, die dabei helfen soll während der Projektlaufzeit Features zu entwickeln und zu priorisieren. • übergeordnete Ziel, identifiziert Bedürfnisse von Zielgruppen, nennt die Kernfeatures, Abgrenzung zur Konkurrenz • Produkt in einem Satz beschreiben. • Eine gemeinsame Sicht auf das große ganze und die unterschiede zur konkurrenz. die schriftlich visualisierte form des eigtl. produktes
2	Haben Sie schon einmal in einem Elevator Pitch eine Produktvision erstellt?  Ja **10** 63% Nein **6** 38% Weiß ich nicht **0** 0%

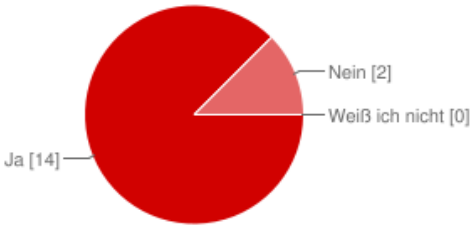
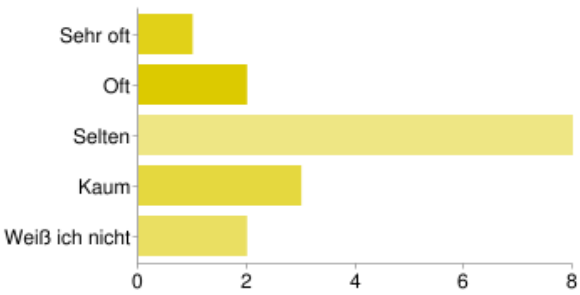
Nr.	Fallstudie: Agile Design - Tools (Produktvision)
3	Wurde die Vision während der Projektlaufzeit reflektiert?  Ja 14 88% Nein 1 6% Weiß ich nicht 1 6%
4	Wurde die Vision während der Projektlaufzeit angepasst?  Ja 9 56% Nein 6 38% Weiß ich nicht 1 6%
5	“Die Produktvision hat am Beginn des Projektes dazu beigetragen, eine klarere Vorstellung vom geplanten Endprodukt zu haben.”  Stimme voll zu 3 19% Stimme eher zu 11 69% Eher nicht 1 6% Gar nicht 0 0% Weiß ich nicht 1 6%

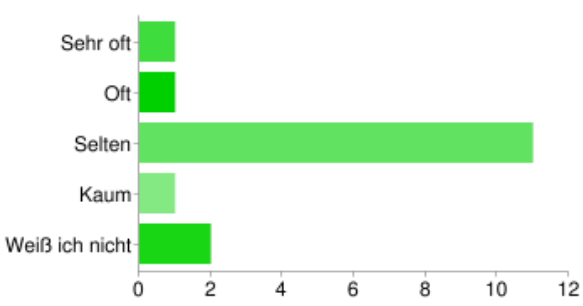
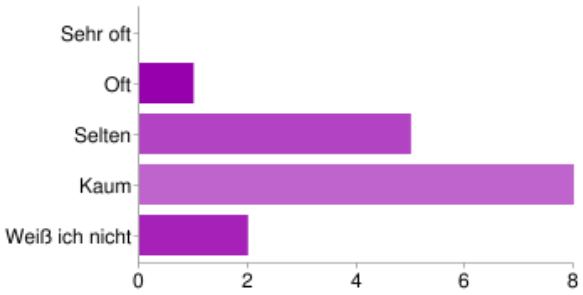
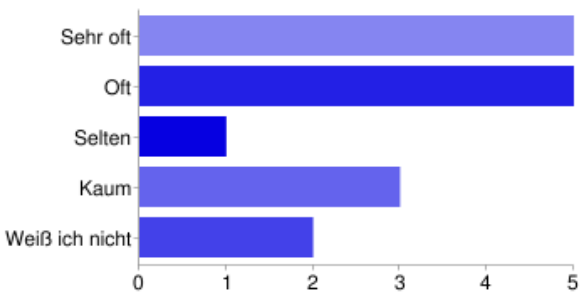
Nr.	Fallstudie: Agile Design - Tools (Produktvision)
6	"Die Produktvision war deutlich und verständlich formuliert."  Stimme voll zu 3 19% Stimme eher zu 9 56% Eher nicht 3 19% Gar nicht 0 0% Weiß ich nicht 1 6%
7	"Die Produktvision war kurz und bündig formuliert."  Stimme voll zu 1 6% Stimme eher zu 13 81% Eher nicht 1 6% Gar nicht 0 0% Weiß ich nicht 1 6%
8	"Die Produktvision hat dazu beigetragen, dass Entscheidungen leichter und effektiver gefällt werden konnten."  Stimme voll zu 2 13% Stimme eher zu 9 56% Eher nicht 4 25% Gar nicht 0 0% Weiß ich nicht 1 6%

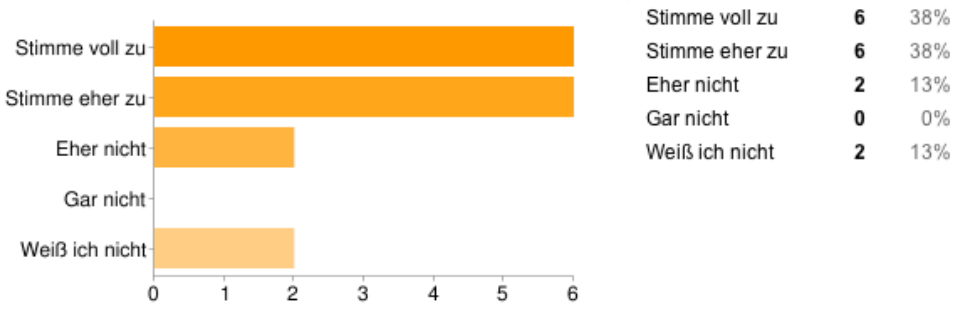
Nr.	Fallstudie: Agile Design - Tools (Styleguide)
1	Was verstehen Sie unter einem interaktiven Styleguide? • Anleitung für Stil, Design und Layout. • Ein Implementierungs-Artefakt, dass die wesentlichen Design-Elemente/Bausteine einer Anwendung anfassbar macht und zu dem den aktuellen Stand der Software bzgl Design wiedergibt • Zusammenfassung aller genutzten Objekttypen an einem Ort mit der Möglichkeit, hier direkt Anpassungen vorzunehmen, mit direkten Auswirkungen auf die Applikation. • styles nicht nur visuell sondern in css code zu definieren • Eine Styleguide, der fortlaufend entwickelt/angepasst wird • stets aktueller HTML-/CSS- bzw. HAML-/SASS-Beispielkatalog der verwendeten Styles, nicht separat vom Produkt selbst gepflegt • Vorgabe für Layout und Design. • Eine Sammlung aller Stilelemente, die sich ständig mit der Anwendung wieterentwickelt. Änderungen haben unmittelbare Auswirkung auf das Design. • HTML/CSS Standartkomponenten-Bibliothek die sich kontinuierlich weiterentwickelt. • Ein interaktiver Styleguide ist ein Styleguide, der nicht zeigt, was wird, sondern was ist. Er tut das, indem er die tatsächlich verwendeten Stilentscheidungen der Anwendung nach Zweck geordnet partitioniert darstellt und übersichtlich präsentiert. • Alle Designelemente werden, bevor sie produktiv eingesetzt werden, vorher auf einer Styleguide Seite implementiert und vom Produktowner abgenommen. • Anleitung zur Darstellung der Marke, die insbesondere die Technik und das Design in Interaktion beschreibt. • HTML, CSS, Klickbar • Ein Implementierungs-Artefakt, dass die wesentlichen Design-Elemente/Bausteine einer Anwendung anfassbar macht und zu dem den aktuellen Stand der Software bzgl Design wiedergibt • Anleitung für Stil, Design und Layout. • technische schnittstelle zwischen entwickler und designer, komponenten bibliothek für entwickler
2	Haben Sie bereits in anderen Projekten mit einer interaktiven HTML Bibliothek zu tun?  Ja 11 69% Nein 2 13% Weiß ich nicht 3 19%

Nr.	Fallstudie: Agile Design - Tools (Styleguide)
3	“Der Styleguide wurde regelmässig refactored.” 
4	“Der Styleguide wurde regelmäßig auf Vollständigkeit überprüft und gepflegt.” 
5	“Der Styleguide eignet sich als technologische Schnittstelle zur Kommunikation zwischen Design und Technik.” 

Nr.	Fallstudie: Agile Design - Tools (Styleguide)
6	“Die Hürde zwischen Entwickler und Designer, also zwischen interaktivem HTML und statischen Grafik-Dateien konnte durch den Styleguide gesenkt werden.”  Stimme voll zu 2 13% Stimme eher zu 5 31% Eher nicht 3 19% Gar nicht 0 0% Weiß ich nicht 6 38%
7	“Der Designer muss in einem agilen Projekt technisches Know-how, z.B. HTML- und CSS-Kenntnisse haben.”  Stimme voll zu 6 38% Stimme eher zu 7 44% Eher nicht 3 19% Gar nicht 0 0% Weiß ich nicht 0 0%
8	Ist solches technisches Wissen der Designer ihrer Erfahrung nach vorhanden?  Sehr oft 0 0% Oft 0 0% Selten 12 75% Kaum 2 13% Weiß ich nicht 2 13%

Nr.	Fallstudie: Agile Design - Tools (Paper Prototyping)
1	Was verstehen Sie unter Papierprototypen? • Skizze der GUI-Komponenten auf Papier • Interaktionsdesign mit Papier testen • Abbildung und Test geplanter Interaktion mithilfe eines Papiermodells • ein papiermodell der seite • komplexe Interaktionsszenarien mit Papier, Stift & Schere simuliert • Layout/Design auf Papier-Elementen. • Abbildung und Test von Funktionalitäten auf Papier • Einfach und schnell zu Lösungen von einfachen Workflows mittels Papier und Stift kommen. • Das, was Carolyn Snyder darüber schreibt. • Programmieren mit Papier • Mockups etc. des Prototypen auf Papier geschrieben gemalt, die die Grundstruktur und Funktionalität des Produkts erklären und durch entsprechende Inhalte "ausprobiert" werden können. • Interaktives Designen von Interfaces mit Hilfe von Papier, an der alle Stakeholder des Projekte teilnehmen können. • Interaktionsdesign mit Papier testen • Skizze der GUI-Komponenten auf Papier • schnelle und kostengünstige weise features zu überprüfen und die interaktion in einer anwendung zu visualisieren ohne sie schon zu implementieren
2	Haben Sie schon einmal in Ihren Projekten Papierprototypen angewandt?  Ja 14 88% Nein 2 13% Weiß ich nicht 0 0%
3	Wie oft wurde Paper Prototyping im Projekt angewandt?  Sehr oft 1 6% Oft 2 13% Selten 8 50% Kaum 3 19% Weiß ich nicht 2 13%

Nr.	Fallstudie: Agile Design - Tools (Paper Prototyping)
4	Wurden während der Konzeptionsmeetings Papierprototypen erstellt?  Sehr oft 1 6% Oft 1 6% Selten 11 69% Kaum 1 6% Weiß ich nicht 2 13%
5	Kamen die erstellten Papierprototypen auch in Benutzertests zum Einsatz?  Sehr oft 0 0% Oft 1 6% Selten 5 31% Kaum 8 50% Weiß ich nicht 2 13%
6	Falls ja: Wurde das Feedback aus den Benutzertests in der Entwicklung und dem weiteren Projektverlauf berücksichtigt?  Sehr oft 5 31% Oft 5 31% Selten 1 6% Kaum 3 19% Weiß ich nicht 2 13%

Nr.	Fallstudie: Agile Design - Tools (Paper Prototyping)
7	"Das Erstellen von Papierprototypen hat zur Verbesserung der Kommunikation beigetragen." 
8	Welche Vorteile ergeben sich ihrer Meinung nach durch das Erstellen von Papierprototypen? • Schneller, einfacher GUI-Test, Schneller, einfacher GUI-Test • Design wird sehr früh anfassbar gemacht. Ist leicht zu ändern. beschränkt sich auf das wesentliche, d.h. zunächst nur die Interaktions und Informations-Strukturierung • Sehr günstige und schnelle Möglichkeit "neue" Interaktionskonzepte zu testen • frühes Feedback, Freiraum für Experimente, geteilte Sicht • Einfache und schnelle Änderung der Desings mit schnellen Ergebnissen. • Vermeidung von Entwicklungsaufwand, Überprüfung der Akzeptanz von Funktionalitäten • Schnell, Einfach, Gemeinsame Sicht auf das Problem und der Lösung • Definition einer gemeinsamen Sprache zwischen Entwicklern, Designern und Stakeholdern, Herabsetzung der Hürden um Ideen auszuprobieren, Adäquate Darstellung von Interaktion • Struktur, Visualisierung, Schnelligkeit, spart Zeit und Kosten • PP hilft dabei eine gemeinsame Sicht auf das Produkt und das Interaktionsmodell zu finden. • Design wird sehr früh anfassbar gemacht. Ist leicht zu ändern. beschränkt sich auf das wesentliche, d.h. zunächst nur die Interaktions und Informations-Strukturierung • interaktion wird deutlich, interface aufwand, usability kann schnell überprüft werden
9	Welche Nachteile ergeben sich ihrer Meinung nach durch das Erstellen von Papierprototypen? • In kurzen Projekten zu viel Overhead. Papierprototypen zu erstellen braucht seine Zeit. Nicht alles muss gepaperprototyped werden. Ziel eines muss festgelegt werden. • keine, wenn einem klar ist, dass auch die Erstellung von Paper-Prototyping mit Aufwand verbunden ist und nach wie vor Abweichungen zum Endprodukt vorhanden sein werden • Papierprototyp hat trotz allem großen Unterschied zum Screen Mockup. • Nervige Bastelei, Bedingt einsetzbar • Geht nur bei einfachen Workflows • Kann von Entwicklern als minderwertige Arbeit wahrgenommen werden, Kann lächerlich gemacht werden • schlechte Lesbarkeit, schlechte Wiederverwendbarkeit, schlechte Weiterverarbeitung • Aufwändig, Konsistenz über die Zeit schwierig, Veraltet schnell • In kurzen Projekten zu viel Overhead. Papierprototypen zu erstellen braucht seine Zeit. Nicht alles muss gepaperprototyped werden. Ziel muss festgelegt werden.

Literaturverzeichnis

[Blu 2011] *Blueprint CSS Grid*. http://blueprintcss.org. Version: 2011. – gesehen am: 16.12.2011 15:00

[it agile 2011] AGILE it: *Was ist agile Softwareentwicklung?* http://www.it-agile.de/wasistagilesoftwareentwicklung.html. Version: 2011. – gesehen am: 17.12.2011 18:00

[Beltrame 2011] BELTRAME, Memi: *Just Married: User Centered Design and Agile*. http://www.slideshare.net/memibeltrame/just-married-user-centered-design-and-agile. Version: 2011. – gesehen am: 20.11.2011 14:30

[Buchner 2008] BUCHNER, Hedi: *Die Produkt-Vision als Leitfaden eines Projekts*. http://www.microtool.de/blog/post/Die-Produkt-Vision-als-Leitfaden-eines-Projektes.aspx. Version: 2008. – gesehen am: 05.12.2011 18:00

[Cecil 2006] CECIL, Richard F.: *Clash of the Titans: Agile and UCD*. http://www.uxmatters.com/mt/archives/2006/12/clash-of-the-titans-agile-and-ucd.php. Version: 2006. – gesehen am: 16.11.2011 10:00

[Chaffee u. Chicoine 2007] CHAFFEE, Alex ; CHICOINE, Leslie: *Are Agile Projects Doomed to Half-Baked Design*. http://www.slideshare.net/theinfonaut/are-agile-projects-doomed-to-halfbaked-design. Version: 2007. – gesehen am: 15.11.2011 12:00

[Cockburn 2007] COCKBURN, Alistair: *Agile software development - the cooperative game*. 2nd ed. München : Addison-Wesley, 2007. – ISBN 0321482751

[Cohn 2010] COHN, Mike: *Agile Softwareentwicklung - Mit Scrum zum Erfolg!* 1. Aufl. München : Pearson Deutschland GmbH, 2010. – ISBN 3827329876

[CSS-Framework 2011] CSS-FRAMEWORK, Compass: *Compass CSS-Framework*. http://compass-style.org. Version: 2011. – gesehen am: 18.12.2011 18:00

[Fox u. a. 2008] Fox, David ; Sillito, Jonathan ; Maurer, Frank: Agile Mthods and User-Centered Design: How these two methodologies are being successfully integrated in industry. In: *Agile 2008 Conference, IEEE Computer Society* (2008), S. 63–72

[Giudice 2008] Giudice, Maria: *Clash of the Titans: Agile and UCD*. http://www.slideshare.net/mgiudice/humancentered-design-meets-agile-development-presentation-625465. Version: 2008. – gesehen am: 16.11.2011 11:00

[HAML 2011] HAML: *HAML*. http://haml-lang.com. Version: 2011. – gesehen am: 18.12.2011 18:00

[Highsmith 2002] Highsmith, Jim: *Agile Software Deleopment Ecosystems*. Boston, Addison Wesley. http://books.google.com/books?id=8Or-q-EXdgEC&printsec=frontcover&hl=de&source=gbs_atb#v=onepage&q&f=false. Version: 2002. – gesehen am: 23.12.2011 17:00

[Hussain u. a. 2009] Hussain, Zahid ; Slany, Wolfgang ; Holzinger, Andreas: Investigating Agile User-Centered Design in Practice: A grounded Theory Perspective. In: *USAB '09 Proceedings of the 5th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society* (2009), S. 279–289

[Maaß u. Savova 2011] Maaß, Dr. C. ; Savova, Elica: *Paper Prototyping in der Softwareentwicklung*. http://www.theseus.joint-research.org/wp-content/uploads/2011/07/paperprototypewisu.pdf. Version: 2011. – gesehen am: 06.12.2011 00:30

[Mayer 2006] Mayer, Marissa A.: *Creativity Loves Constraints*. http://www.businessweek.com/magazine/content/06_07/b3971144.htm. Version: 2006. – gesehen am: 05.12.2011 17:00

[Miller 2005] Miller, Lynn: Case Study of Customer Input For a Successful Product, 2005 (Agile Development Conference), S. 225–234

[Miller u. Sy 2007] Miller, Lynn ; Sy, Desiree: *Summary of the main problems experienced by participants while doing agile UCD*. CHI/UPA Summary. http://www.slideshare.net/abcd82/bringing-usercentereddesign-practices-intoagile-development-projects-presentation. Version: 2007. – gesehen am: 28.11.2011 16:00

[Najafi u. Toyoshiba 2008] Najafi, Maryam ; Toyoshiba, Len: Two Case Studies of User Experience Design and Agile Development. In: *Agile 2008 Conference, IEEE Computer Society* (2008), S. 531–536

[Nebe u. a. 2007] Nebe, Karsten ; Düchting, Markus ; Zimmermann, Dirk: *Integration von User Centred Design Aktivitäten in Agile Softwareentwicklung*. http://factor-u.com/media/UPA07_NebeDuechtingZimmermann.pdf. Version: 2007. – gesehen am: 20.11.2011 18:30

[Obexer 2007] Obexer, Günther: *Paper Prototyping*. http://www.medien.ifi.lmu.de/lehre/ws0607/mmi1/essays/Guenther-Obexer.xhtml. Version: 2007. – gesehen am: 18.12.2011 19:00

[Patton 2009] Patton, James: *Bringing User-centered design practices into agile development projects*. http://www.slideshare.net/abcd82/bringing-usercentereddesign-practices-intoagile-development-projects-presentation. Version: 2009. – gesehen am: 23.12.2011 16:00

[Patton u. Halley 2009] Patton, Jeff ; Halley, Lane: *Making Sense of UCD and Agile*. http://www.slideshare.net/LaneHalley/making-sense-of-ucd-and-agile. Version: 2009. – gesehen am: 15.11.2011 12:00

[Pichler 2009] Pichler, Roman: *The Product Vision*. http://www.scrumalliance.org/articles/115-the-product-vision. Version: 2009. – gesehen am: 05.12.2011 19:00

[Poppendieck 2011] Poppendieck, Mary: *First build the right thing*. http://www.slideshare.net/AgileOnTheBeach/first-build-the-right-thing. Version: 2011. – gesehen am: 17.12.2011 14:30

[Porter 2008] Porter, Joshua: *Designing for the Social Web*. Berkeley, CA, New Riders, 2008. – ISBN 978–0321534927

[Ramsay 2011a] Ramsay, Anders: *Agile UX and The One Change That Changes Everything*. http://www.andersramsay.com/2010/07/08/agile-ux-and-the-one-change-that-changes-everything. Version: 2011. – gesehen am: 20.11.2011 18:30

[Ramsay 2011b] Ramsay, Anders: *Cross-Functional Pairing: What it is and why you should be doing it*. http://www.slideshare.net/andersr/crossfunctional-pairing-what-it-is-and-why-you-should-be-doing-it. Version: 2011. – gesehen am: 10.12.2011 17:30

[Raskin 2007] Raskin, Aza: *Iterative Design: Towards the Perfect Paper Plane*. http://humanized.com/weblog/2007/06/05/iterative_design_isnt_design_by. Version: 2007. – gesehen am: 05.12.2011 16:00

[SASS 2011] SASS: *SASS*. http://sass-lang.com. Version: 2011. – gesehen am: 18.12.2011 18:00

[Scherer 2011] Scherer, Josef: *Scrum Fibel*. http://scrum-fibel.de/artefakte/scrum%20artefakte.html. Version: 2011. – gesehen am: 18.12.2011 20:00

[Schwaber u. Sutherland 2011] Schwaber, Ken ; Sutherland, Jeff: *The Scrum Guide*. http://www.scrum.org/storage/scrumguides/Scrum_Guide.pdf. Version: Oktober 2011. – gesehen am: 20.11.2011 18:00

[Snyder 2003a] Snyder, Carolyn: *Paper Prototyping*. http://www2.engr.arizona.edu/~ece596c/lysecky/uploads/Main/Lec6.pdf. Version: 2003. – gesehen am: 18.12.2011 19:00

[Snyder 2003b] Snyder, Carolyn: *Six Signs That You Should Use Paper Prototyping*. Artikel auf JAVA.net. http://today.java.net/pub/a/today/2003/12/23/sixSigns.html. Version: 2003. – gesehen am: 05.12.2011 22:00

[Stangl 2011] Stangl, Werner: *Fragen und Probleme in Gruppen - Phasen der Gruppenentwicklung*. http://arbeitsblaetter.stangl-taller.at/KOMMUNIKATION/Anfangsprobleme.shtml. Version: 2011. – gesehen am: 17.12.2011 17:00

[Stein 2004] Stein, Sebastian: *Emergenz in der Softwareentwicklung - bereits verwirklicht oder Chance?* http://emergenz.hpfsc.de/html/index.html, FH Dresden, Diplomarbeit Fachbereich Informatik, Mai 2004

[Sy 2007] Sy, Desiree: Adapting Usability Investigations for Agile User-centered Design. In: *Journal of Usability Studies* Vol. 2, Issue 3 (2007), Mai, S. 112–130

[tutor2u 2011] tutor2u: Agile Development Iterations and UI Design. In: *tutor2u - GCSE Business Studies* (2011). http://tutor2u.net/business/gcse/downloads/organisation_aims_objectives.pdf

[Ungar 2008] Ungar, Jim: The Design Studio: Interface Design for Agile Teams. In: *Agile 2008 Conference, IEEE Computer Society* (2008), S. 519–524

[Vogt 2007] Vogt, Ingo: *Agile Softwareentwicklung.* ORDIXNews 2007. http://www.ordix.de/ORDIXNews/3_2007/Projektmanagement/agile_softwareentwicklung.html. Version: 2007. – gesehen am: 15.11.2011 15:00

[Wolf 2007] Wolf, Henning: *Die agile Chance: Flashfilm.* auf der Jax 2007 Konferenz in Wiesbaden. http://www.it-agile.de/agilechanceflash.html. Version: 2007. – gesehen am: 05.11.2011 12:00

[Zuser u. a. 2001] Zuser, Wolfgang ; Biffl, Stefan ; Grechenig, Thomas ; Köhle, Monika: *Software Engineering - mit UML und dem Unified Process.* München, Pearson Studium, 2001. – 45ff S.

Glossar

Agile UCD Bezeichnet normales User-Centered Design unterzuhilfenahme von agilen Methoden. 6, 33

Alphatest Bezeichnet die erste Testphase einer zu testenden Anwendung durch Fremde Benutzer die sich in der Alpha-Version befindet. 33

Anomalien Sind Abweichungen von der Regel oder von üblichen Strukturen. 17

Autocomplete Hierbei handelt es sich um ein Eingabefeld welches dem Benutzer Vorschläge für bestimmte Texteingaben liefert während der Benutzer das Wort eingibt. 77

Betatest Bezeichnet eine Testphase und die Programmfehlersuche einer Anwendung die sich in einem Entwicklungsstadium einer Beta-Version befindet. 33

Blueprint Ist ein relativ verbreitetes CSS-Framework welches viele CSS Komponenten beinhaltet die man direkt in seiner Anwendung benutzen kann. 70

Bottleneck Steht für eine einzelne oder limitierte Anzahl an Komponenten oder Ressourcen die für ein Performanz- oder Kapazitätsmangel eines gesamten Systems verantwortlich sind. 62

Business Value Im Management Kontext ist der Business Value eine formlose Bezeichnung aller möglichen Formen von Wert, die die Gesundheit und das Wohlergehen eines Unternehmens auf lange Sicht definieren. 59

Change Management (deutsch Veränderungsmanagement) - Bezeichnet alle Aufgaben und Maßnahmen die für eine weitreichende Umsetzung von neuen Strategien, Strukturen oder Prozessen in einem Unternehmen eingesetzt werden. 13

Clearfix Ansätze um bekannte Browser Probleme bei fliessenden Elementen zu bereinigen. 67

Cross-Browser-Kompatibilität Die Fähigkeit von Webseiten eine gleiche Ausgabe auf unterschiedlichen Browsertypen zu erzeugen und die sich somit unabhängig davon identisch verhalten. 40

CSS-Hack Es ist möglich, Browsern die gewünschten CSS-Eigenschaften zukommen zu lassen und problematische Deklarationen vor anderen Browsern mit Hilfe einer CSS-Browserweiche zu „verstecken“ - dies wird als „Hack“ bezeichnet. 67

CSS-Selektor Selektoren dienen dazu, bestimmt Elemente in einem HTML/XML-Dokument auszuwählen. 65

Deadline Ein festgelegter Abgabetermin zu dem eine bestimmte Aufgabe erledigt werden soll. 23

Definition of Done Hierbei handelt es sich um eine einfache Liste von Aktivitäten die zu einem allgemeinen Verständnis sorgen sollen wann eine Implementierte Funktionalität abgeschlossen ist und nachweisbar Wert für das Produkt liefert. 43

Design by committee Dieser Begriff steht für einen Designprozess bei dem die Entscheidung für ein bestimmtes Design, durch mehrere Teilnehmer die evtl. auch nicht das nötige Fachwissen und den Überblick über das Projekt besitzen, getroffen wird. 9

Design-Chunks Design-Chunks sind kleine Designeinheiten die sich während eines Sprints leicht realisieren lassen. Diese Designeinheiten erweitern inkrementell das Gesamtdesign der Anwendung. 8

DRY Don’t Repeat Yourself - Dieses Prinzip besagt, dass weder Daten noch Funktionalität redundant im System vorkommen sollen, da andernfalls der Wartungsaufwand erhöht wird. 63, 64

empirischer Prozesssteuerung Beschreibt das bei einem Produkt welches an Qualität verliert Nachbesserungen durchgeführt werden müssen, diese stützen sich auf den drei Säulen der Sichtbarkeit, Inspektion und Anpassung. 17

Epic Steht für eine grob formulierte Anforderung an eine Anwendung im Kontext des Anforderungsmanagements in der agilen Softwareentwicklung. 58

explorativen Besagt das jemand nur geringe Kenntnisse über etwas hat und sich mit einem Thema quasi Versuchsweise beschäftigt um Erfahrungen darin zu sammeln. 53

explorativer Besagt das sich jemand mit etwas beschäftigt über das er bisher nur geringe Kenntnisse besitzt. 24

Extreme Programming Extreme Programming - Beschreibt ein Vorgehen der Softwaretechnik bei dem das Lösen einer Aufgabe in den Vordergrund gestellt wird indem man sich den Anforderungen des Kunden in kleinen Schritten annähert. 13

Fancy-Buttons Ruby gem zum Download, url: `https://github.com/imathis/fancy-buttons`. 76

Feature Umfasst einen bestimmten Teil einer Funktionalität der Anwendung. 43

Grid-Layout Ist eine Struktur die als Grundlage der Layouterstellung genommen wird, sie ordnet Komponenten in einem Gitter an, das aus gleich großen Gitterzellen besteht. 70, 73

HAML Steht für **H**TML **A**bstraction **M**arkup **L**anguage und ist eine vereinfachte Auszeichnungssprache die für eine klare und einfache Art das XHTML benutzt wird. 63

HTML-Mockup Hierbei handelt es sich um ein grobes Model eines Designs oder einer Anwendung ohne Funktionalität welches zum lernen, demonstrieren oder evaluieren benutzt wird. 36, 37, 45

JAVA Ist eine objektorientierte Programmiersprache und Warenzeichen von Sun Microsystems. 30

Markup-Sprache Eine Auszeichnungssprache (englisch: Markup Language) dient zur Beschreibung des Inhalts eines Dokumentes und teilweise zur Beschreibung des Verfahrens, welches zur Bearbeitung dieser Daten benötigt wird. 63

Photoshop Eine Bildbearbeitungssoftware des Adobe Systems Unternehmens aus den USA. Dieses Programm ist in diesem Bereich Marktführer und wird fast überall im professionellen Bereich zur Bildbearbeitung eingesetzt. 30

pitchen Auch bekannt unter Agenturpitch, bezeichnet den Wettbewerb unterschiedlicher Agenturen um einen bestimmtes Budget eines Unternehmens für dessen Betreuung zu gewinnen. 23

Rails-Helper Das sind Hilfsklassen die kleine Funktionen beinhalten die sich nur mit der Ausgabe auf der Oberfläche beschäftigen und die zu komplex für eine HTML Datei sind. 73

Refactoring Refactoring ist eine Technik, bei der die innere Struktur verbessert wird, ohne dabei das Verhalten nach außen hin zu verändern. Sie erfolgt in kleinen, mechanischen Schritten, die zusammengenommen den Code signifikant verbessern können. 41

Reset-Style Ansatz der die Basisstyles des Browsers zurückzusetzt um eine einheitliche Umgebung zu schaffen. 67

Risikomanagement Bezeichnet ein systematisches Verfahren zur Erfassung und Bewertung von Risiken in einem Unternehmen. 13

Ruby Ist eine in den 90er Jahren entworfene höhere Programmiersprache, sie ist interpretiert und objektorientiert. 30

Ruby-Gem Ist eine gepackte Ruby Applikation oder Bibliothek. 76

Ruby-on-Rails Oft auch nur Rails genannt ist ein auf der Programmiersprache Ruby basierendes Web Applikations Framework. 64

SASS Steht für Syntactically Awesome Stylesheets und ist Meta-Sprache für CSS, sie abstrahiert den CSS-Code um einfachere Stylesheet-Dateien zu kreieren. 63

Scribble Hierbei handelt es sich um grob skizzierte Abbildungen der späteren Anwendung und den Aufbau der auf den einzelnen Seiten befindlichen Elemente. 45

simultan Steht für Gleichzeitigkeit oder auch Zeitgleich. 8

Sprint-Zero Planung und kontextbezogene Recherche von relevanten Daten für die bevorstehenden Sprints. 7, 8

Sprite Steht für eine Ansammlung von unterschiedlichen Grafiken in einer einzigen Datei, die einzelnen Grafiken werden dann durch bestimmte Positionierung der Hauptdatei auf der Web-Seite angezeigt. 67

Templating Templating beschreibt die Benutzung von Vorlagen, auf deren Grundlage die Gestaltung anderer Seiten durchgeführt wird. 24

Timebox Eine Timebox ist ein fest vorgegebener Zeitrahmen für verschiedene Vorgänge im Projekt. 18, 20, 21

UCD User-Centered Design ist eine Design-Philosophie und ein Prozess welcher die Bedürfnisse, Wünsche und Einschränkungen von Benutzern einer Anwendung an jeder Stelle des Prozesses in den Vordergrund stellt. 5, 8, 32

Usability-Testing Ein Usability-Test wird durchgeführt, um die Benutzbarkeit einer Anwendung durch Überprüfung der in Frage kommenden Benutzer zu gewehrleisten. 5

User Research Einbeziehung der Benutzergruppe in die konzeptionelle Phase. Gemeinsame Entwicklung von Prototypen und weitere konzeptuelle Tätigkeiten. 32

User Story User Stories werden im Rahmen der Agilen Softwareentwicklung zusammen mit Akzeptanz-Kriterien zur Spezifikation von Anforderungen einer Anwendung verwendet. 43, 58

W3C Das World Wide Web Consortium ist ein Gremium welches sich mit der Standardisierung der im Internet betreffenden Techniken beschäftigt. 68

Webstandard Begriff über formale Standards und andere technische Spezigfikationen die Aspekte des World Wide Web definieren und beschreiben. 40

Wireframe Ähnlich wie Mockups sind hier schematische Darstellungen einzelner Seitenvorlage gemeint die grundlegenden Elemente der Seite festhalten. 45

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 10. Februar 2012 Gregor Swatek