Agile Softwareentwicklung

SEMINAR „SOFTWAREENTWICKLUNG IN DER WISSENSCHAFT“

ROBERTO SEIDEL (BETREUER: DR. JULIAN KUNKEL)

Agenda

1. Motivation: Die Gründe für Agile Softwareentwicklung

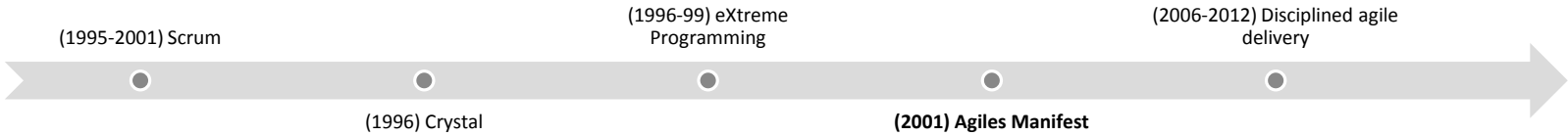
2. Grundlagen der agilen Softwareentwicklung
3. Framework: Scrum
4. Framework: eXtreme Programming (kurz XP) – inkl. Studien zum Nutzen

5. Zusammenfassung / Empfehlungen

1. Motivation

GRÜNDE FÜR FRAMEWORKS UND AGILE SOFTWAREENTWICKLUNG

Historischer Ablauf



1. Verschiedene „leichtgewichtige“ Methoden zur Softwareentwicklung in der Praxis entwickelt

2. Treffen von 17 Methoden-Entwickler & Interessierten
   - „Agile Allianz“ gebildet
   - „Agiles Manifest“ erarbeitet (Werte, gemeinsame Prinzipien)

3. Anpassungen alter Frameworks an neue Erkenntnisse
   Entwicklung neuer Frameworks (z.T. mit beabsichtigten Abweichungen vom agilen Manifest)

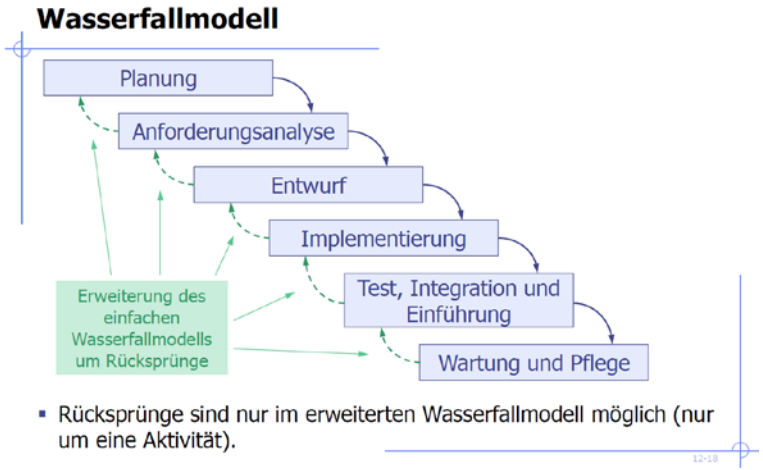
Gründe für Frameworks

Zuverlässigkeit gewährleisten

Entwicklerteams koordinieren

➔ Vorgehensmodelle vor agiler Softwareentwicklung:

- Wasserfallmodelle
- V-Modell (ursprüngliche Variante)
- Unified Process
- Spiralmodell



aus: Dr. G. Schneidereit (Uni Hamburg); 2013/14; Modul Grundlagen der Wirtschaftsinformatik Foliensatz 12 Folie 18

Warum Agile Softwareentwicklung?

Probleme traditioneller Softwareentwicklung:

- stark reguliert, reglementiert, im Detail durchorganisiert
- dokumentationsgetrieben
- „design for the future“
- über Arbeitsweise & Personaleinsatz entscheiden Manager außerhalb der Entwicklungsteams

Schwierigkeiten der Softwareentwicklung allgemein:

- komplexe Probleme
- verschiedene Interessensgruppen
- schnell ändernde(s) Umfeld + Anforderungen

➔flexibleres Vorgehen nötig

➔empirisches Vorgehen empfohlen

2. Grundlagen agiler Softwareentwicklung

Grundlagen agiler Softwareentwicklung

Gegenkonzept zu linearer Entwicklung und starren Frameworks

Idee: möglichst einfache & bewegliche Teilprozesse

Hauptziele: „Kunden“ zufriedenstellen, schnellerer Einsatz der entwickelten Systeme

Werte agiler Softwareentwicklung:

◦ **Individuen und Interaktionen**	über	*Prozessen und Werkzeugen*
◦ **Funktionierende Software**	über	*umfassende Dokumentation*
◦ **Zusammenarbeit mit dem Kunden**	über	*Vertragsverhandlung*
◦ **Reagieren auf Veränderung**	über	*das Befolgen eines Plans*

(nach http://agilemanifesto.org/, Kent Beck et al.)

→ Weniger Overhead, Menschen im Mittelpunkt

Prinzipien

auf selbstorganisierte Teams vertrauen

Zusammenarbeit von motivierten Fachexperten & Entwicklern in konstanter Intensität

Informationsaustausch durch persönliche Gespräche

funktionierende Software in kurzen Zyklen herausgeben

Offenheit für Anforderungsänderungen

technisch einfache & exzellente Lösungen anstreben

regelmäßige Reflexion & Verhalten anpassen

Eigenschaften

Annahmen

- Anforderungen & Vorgaben sind anfangs unvollständig/unklar oder ändern sich während des Projekts

Methoden & Elemente

- iteratives Vorgehen bei allen Aufgaben, inkrementelle Entwicklung
- regelmäßige Reflexion
- frühes Implementieren & Release
- regelmäßiges Testen während der Entwicklung

Grenzen des Einsatzgebiets

stabile & vollständige Anforderung ➔ geringe/keine Vorteile

räumliche Nähe der Entwickler notwendig (globale Forscherteams? Open Source Projekte?)

Akzeptanz veränderlicher Kundenanforderungen→ Konfliktpotenzial

Angebote mit Festpreis schwierig:

- aufgrund von Anforderungsänderungen sind Preis / Gesamtdauer anfangs u.U. nicht abschätzbar
- (XP Empfehlung: Pay-per-Use)

Vorteile

frühe(r) Nutzen/Erfolgserlebnisse durch Produktinkremente & Anforderungspriorisierung

vollständiger Projektabbruch + Neustart bei stark veränderter Zielsetzung nicht unbedingt nötig
➔ kein Totalverlust bisheriger Investitionen

stärkere Einbeziehung von Kunden und ggf. Anwender kann Akzeptanz bei Einführung steigern

3. SCRUM

Scrum

1995 von Jeff Sutherland & Ken Schwaber publiziert

Heute: verbreitetstes Framework zur Softwareentwicklung*[1][2]*

➔ Risiko minimieren & Wert steigern

verschiedene Nutzungsszenarien möglich

Zentrale Komponenten:

- **selbstständige, unabhängige Entwicklungsteams (bzgl. Fähigkeiten & Befugnissen)**
- Transparenz
- Regelmäßige Überprüfung & Anpassung

Idee

empirische Prozesssteuerung: Entscheidung anhand von Erfahrungen

iteratives Vorgehen, inkrementelle Entwicklung

viele formalisierte Elemente statt eines festen Prozesses

- Ereignisse (Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospektive)
- Artefakte (Product Backlog, Sprint Backlog, Definition of Done)

formelle Kommunikation zur Strukturierung + informelle Kommunikation (kreativer, schneller)

Qualitätsanspruch soll steigen, nie sinken (bei Überschätzung: Leistungs-/Arbeitsumfang senken)

Rollen im Scrum-Team

Scrum Master („Coach“):

- Idee von Scrum vermitteln
- Einhaltung von Theorie, Praktiken, Regeln durchsetzen
- Kommunikation mit Außenstehenden effizient gestalten

Product Owner (einzelne Person):

- Nutzen der Arbeit und des Produkts maximieren
- Product Backlog (Aufgaben, Funktionsanforderungen) managen → alleinverantwortlich!
  - verständlich, transparent, sinnvoll sortiert/priorisiert

➢steuert als einziger Arbeit des Teams

Entwicklungsteam (interdisziplinäres Team):

- erstellen Produkt(inkremente)
- keine weitere Unterteilung / Hierarchie
- gemeinsam für Umsetzung verantwortlich (auch außerhalb des individuellen Spezialgebiets)

Projektablauf

Product Backlock anlegen

mehrere Iterationen = „Sprints“ (maximal 1 Monat)

- definierter Leistungsumfang
- potenziell auslieferbares Produkt-Inkrement als Ziel

1. Sprint Planning (max. 8 Std.) ➔ Sprint Backlog anlegen
   - Was wird im Inkrement umgesetzt?
   - Verständnis über Arbeitsinhalte erarbeiten
   - Sprint-Ziel formulieren: Zweck des Inkrements
   - Wie wird das Ziel erreicht? *(nur Entwicklungsteam)*
   - Arbeiten der ersten Tage in kleine Einheiten zerlegen

Projektablauf

2. Daily Scrums – nur Entwicklungsteam (bis zu 15 Min.)
   - überprüft Arbeit seit letztem Daily Scrum
   - prognostiziert Ergebnisse bis zum nächsten Daily Scrum
   - identifiziert ggf. Hindernisse
   - informelle Treffen für detaillierte Diskussionen und Umplanungen im Anschluss
3. Sprint Review – mit Stakeholdern (max. 4 Std.)
   - Vorführung des Inkrements + Fragen beantworten (Feedbackmöglichkeit)
   - Ggf. Probleme inkl. Lösung schildern
   - Anpassung des Product Backlogs, neue Prioritäten
   - Aktualisierung der Prognose zur Fertigstellung
4. Sprint Retrospektive (max. 3 Std.)
   - Einsatz der Beteiligten, Beziehungen, Prozesse & Werkzeuge überprüfen
   - Erfolge & mögliche Verbesserungen identifizieren
   - Verbesserung von Arbeitsweise & Produktqualität planen

Spezifische Vorteile & Grenzen

ermöglicht Reaktion auf Änderungen bei Anforderungen/Technologie ➔ Komplexitätsreduktion

- Iterationen, Reflexion & Anpassung

kein Automatismus für Agilität & Qualität

- Erkenntnisse müssen genutzt werden
- Bereitschaft zur Selbstorganisation nötig

Transparenz essentiell wichtig, aber nicht immer vollständig möglich

4. eXtreme Programming

GRUNDLAGEN DES FRAMEWORKS

eXtreme Programming *(kurz: XP)*

1999 von Kent Beck publiziert, 2004 weiterentwickelt

will technische und soziale Aspekte verbessern

abgeleitet aus Best Practices, z.B. Test Driven Development (1960 NASA)

verglichen mit Scrum: konkretere, detailliertere Vorgaben zur Methodik

zentrale Werte: Kommunikation, Einfachheit, Feedback, Mut (Ehrlichkeit), Respekt

<u>Ein Kerngedanke:</u>

„Do your best and then deal with the consequences. That‘s extreme.“ *- [Beck2005], Seite 1*

→ Risiken zur Minimierung aktiv angehen (Prinzip: Fehlschläge hinnehmen)

Eigenschaften & Praktiken

User Stories (Handlungen die ermöglicht werden sollen) repräsentieren Anforderungen

Themes vermitteln Gesamtbild

Planning-Game/-Poker:

- Team + Kunde schätzen Aufwand der User Stories, spezifizieren nächste Version

Beschränkung auf wirklich notwendige Features

Überstunden vermeiden ➔ konstante, nachhaltige Leistung fördern

informativer, gemeinsamer Arbeitsplatz

Eigenschaften & Praktiken

frühe Unit-Tests / Test Driven Development

- ➔ späte aufwendige Korrekturen vermeiden

automatisierte Integration in Gesamtsystem + Regressionstest (10-Minute Build)

- ➔ frühe Fehlererkennung, Austausch von Zwischenergebnissen

kollektives Eigentum, kollektive Verantwortung, gemeinsame Code-Basis

- ➔ jeder soll sich dem Projekt verpflichtet fühlen

Rollen im Team

Entwicklerteam

- keine feste Unterteilung, stattdessen Wissensaustausch & situationsabhängige Aufgabenverteilung
  - ➔ Folgen eines Personalausfalls mindern,
  - ➔ einzelne Entwickler entlasten

Product Owner (z.B. Kunde, Nutzer oder Produktmanagement Vertreter)

- verantwortlich für Projekterfolg
- priorisiert Aufgaben
- entscheidet über Vorgehen („Wie?“)

Kunde

- kontinuierlich vor Ort, aktive Mitarbeit
- darf Zwischenversionen sehen & jederzeit Änderungswünsche äußern
- entscheidet über Features/Anforderungen („Was?“)

→ engere Einbindung als bei Scrum (kontinuierlich vor Ort vs. Sprint Reviews)

➔ bedarfsgerechte Produkte, keine Überraschungen

Nutzer

(Teammanager)

- „disziplinarischer Vorgesetzter“ *(aus [wiki1])*
- geringe Relevanz für XP

Projektablauf

Anforderungsbeschreibung:

- Keine Projektspezifikation am Anfang! ➔ aufwändig, hohe Kosten bei späteren Änderungen
- Kunde beschreibt mit dem Team die **Anforderungen durch User Stories**
- Class Responsibility Collaboration Model **(CRC) Cards** können gesamte **Anwendersicht verdeutlichen**

Planung:

- Aufwandsabschätzung der User Stories mit Story Points im Planning-Game
- Priorisierung (zuerst: höchster Nutzen & höchstes Risiko, danach geringes Risiko)
- **Release-Plan** bestehend aus Iterationen (quartalsweise)

  ➔ **Iteration** bestehend aus User Storys (wöchentlich)

  ➔ **User Story** wird in **mehreren Aufgaben** geteilt, Aufwandsschätzung der Aufgaben

Projektablauf

Umsetzung von Iterationen des Release-Plans:

- Entwickler nehmen sich nach und nach Aufgaben(-pakete)
- Akzeptanztests durch Kunden im Abstand einiger Tagen
- Stand Up Meeting täglich (vgl. Daily Scrum)
- Arbeitspaare situationsbedingt bilden (Pair-Programming)
- Modultest implementieren (möglichst automatisiert)
- Funktionalität implementieren
- Refactoring neben Implementation

Begutachtung/Rückmeldungen durch Stakeholder (Management, Kunde,…) nach jeder Iteration

Erkenntnisgewinn oder veränderte Prioritäten des Kunden**kann während der Iteration** zur Änderung der User Story Auswahl führen

Release nur bei ausreichendem, geschlossenem neuen Funktionsumfang

4. eXtreme Programming

STUDIEN ZUM PRAXISEINSATZ

Studien zum Nutzen

Befragung zu Erfahrungen mit XP Projekten

45 Teilnehmer

unterschiedliche Rollen im Team, Projektarten, Firmen-/Teamgrößen, Branchen, Länder,…

22 laufende Projekte, 23 abgeschlossene

- großes Wachstum bei der Verbreitung
- 44 Projekte „erfolgreich“ (7-9 auf 9er-Skala)
  1 Projekt „teilweise erfolgreich“
- 45 empfehlen XP aktiv weiter

➔ für innovative Unternehmen geeignet

➔ interessanter Ansatz zur Softwareentwicklung

[Rumpe2002]

Studien zum Nutzen

Fallstudie über Zeitbedarf, Leistung und Qualität

1 Team (4 Entwickler) – 6 Iterationen

Entwicklungszeit: 8 Wochen

Entwicklung eines web-basierten (Java & JSP) Datenmanagementsystems im wissenschaftlichen Kontext

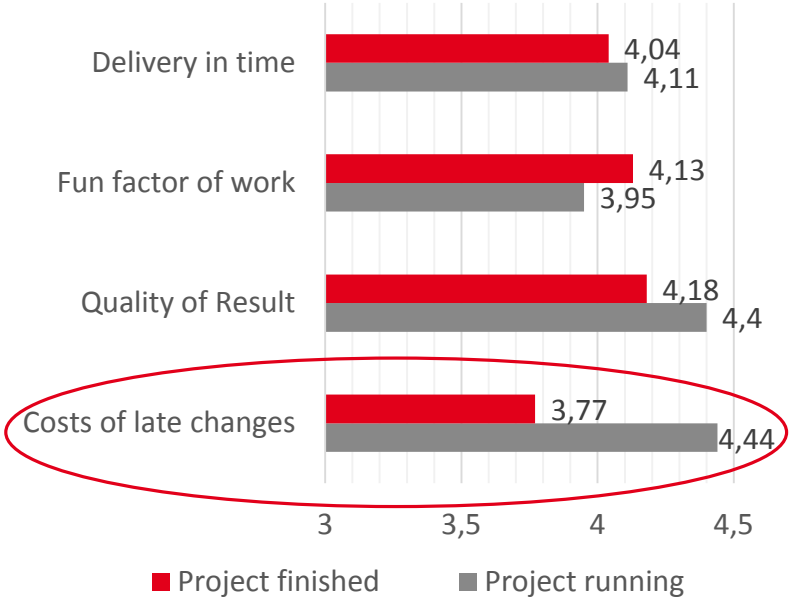
Autoren als Vor-Ort-Kunde, Manager involviert

- nicht alle Prinzipien notwendig:
  Bsp. Kunde vor Ort (21% der Zeit genutzt)
  → Kritikpunkt entkräftet:
  notwendige Umsetzung von „allem oder nichts“
  → Personalkosten geringer als erwartet
  (beim Kunden)

[Abrahamsson2004]

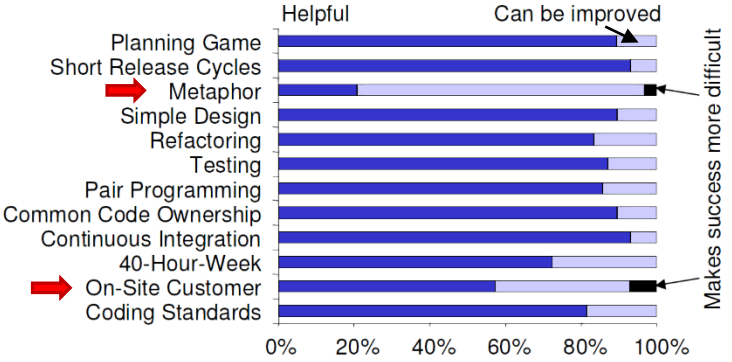
- zu kurze Iterationen senken Produktivität & Qualität
- niedrigerer Anteil der Programmierarbeit (54,7%) als aus der Literatur erwartet
- feinere Aufgabenunterteilung schnell erlernt
  → geringerer Zeitverlust durch fehlerhafte Aufwandsschätzungen
- spezielle Anforderungen leicht integrierbar (Datenerhebung zum Prozess)
- Anwesenheit von Kunde + Anwendern als Motivation & Wertschätzung wahrgenommen
- Verschiebung von Feature-Releases weniger unerwartet für den Kunden

Befragung zum Nutzen – Vorteile



Erreichung ausgewählter XP Ziele im Vergleich zu traditionellen Methoden
(Skala: -5 „deutlich schlechter“ bis 5 „vollständig erreicht“)

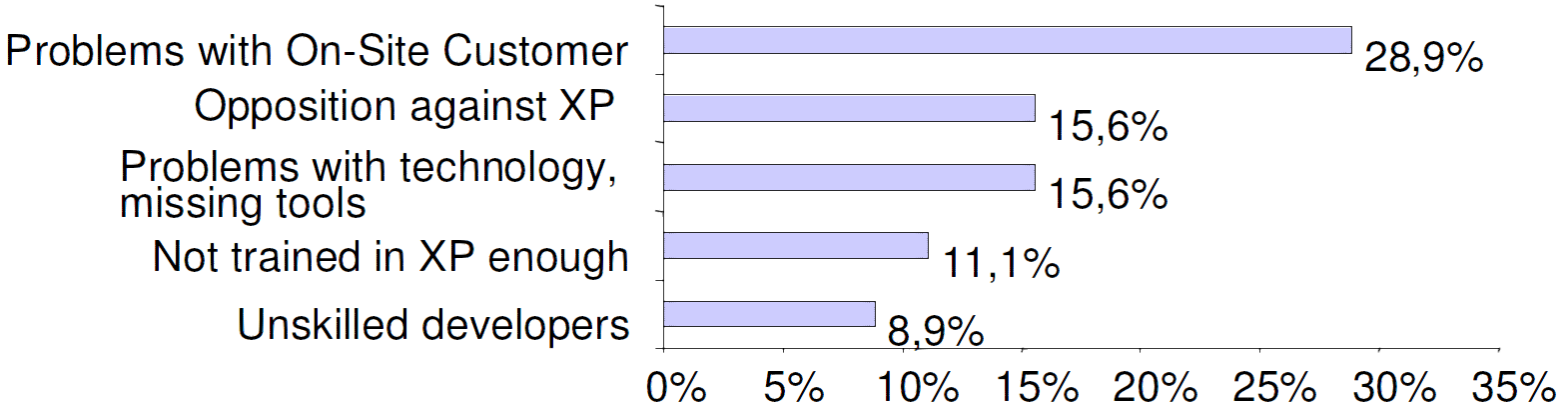
nach: [Rumpe2002], Seite 5



Nutzen einzelner XP Elemente
(Auswahlmöglichkeiten: „helpful, improvable, or even making-difficult”)

aus: [Rumpe2002], Seite 4

Befragung zum Nutzen – Risiken



Größte Risiken für den Projekterfolg
(Freier Text – nachträglich zu Kategorien zusammengefasst)

aus: [Rumpe2002], Seite 5

Fallstudie (Aufwandschätzung)

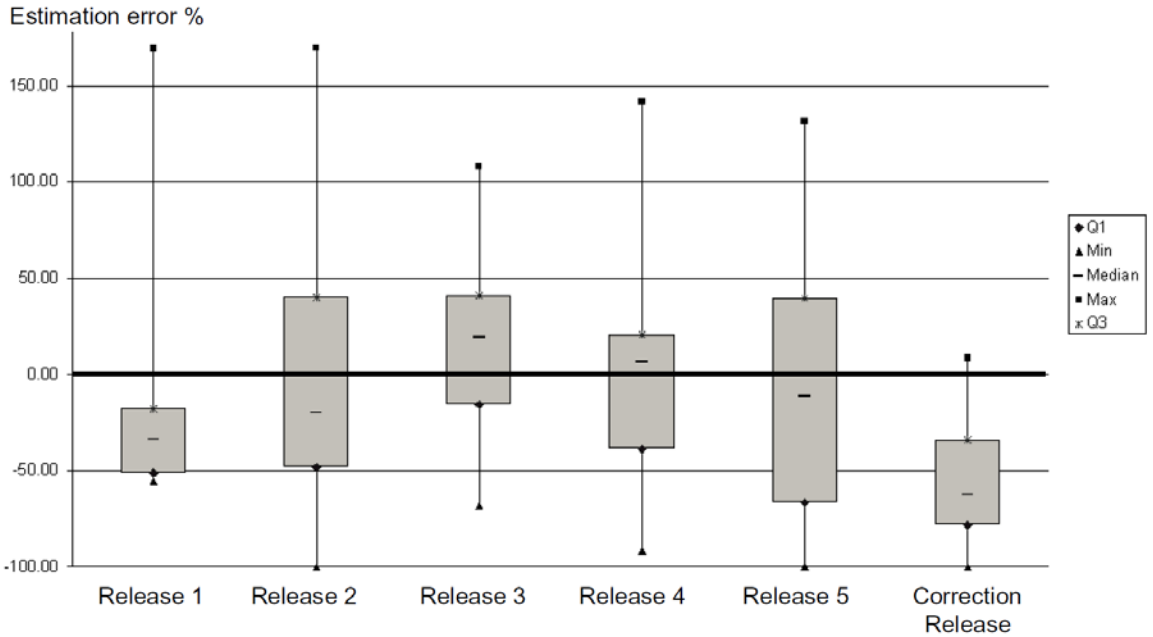


Figure 2. Estimation accuracy

Relative Abweichungen der Aufwandsschätzungen je Iteration

aus: [Abrahamsson2004], Seite 6

anfangs: Tendenz zur Überschätzung des Aufwandes

Median nähert sich 0

Varianz in der Abweichung der Aufwandsschätzung bleibt hoch

Fallstudie (Aufwandsschätzung)

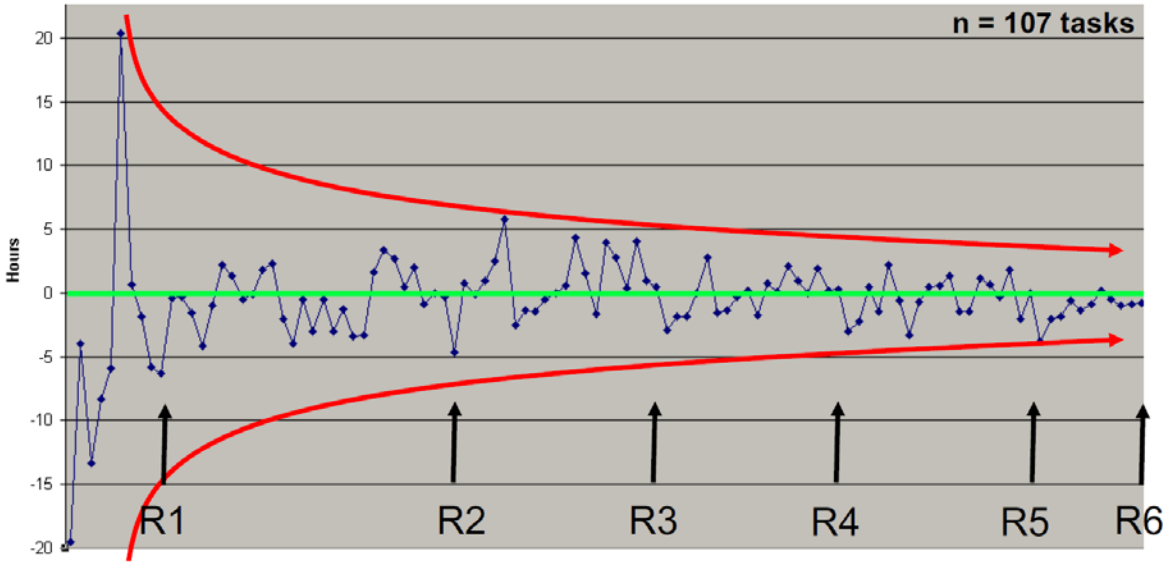


Figure 3. XP Pulse: Hours lost by faulty estimates

Abweichung der Aufwands-schätzung verringert

➔ weniger falsch zugewiesene Ressourcen
➔ geringerer Zeitverlust

<u>hypothetischer Grund:</u>

User Stories in kleinere Aufgaben aufgeteilt

Absolute Abweichungen der Aufwandsschätzungen

aus: [Abrahamsson2004], Seite 6

Fallstudie (Refactoring-Einfluss)

	R1	R2	R3	R4	R5	CR
Productivity (Loc/hour)	13.39	25.12	16.63	9.02	20.05	8.4
Refactoring %	13.8	5.9	18.1	18.0	11.2	0

Table 4. Productivity vs. Refactoring

Möglicher Einfluss des Refactorings auf die Produktivität

aus: [Abrahamsson2004], Seite 8

Fallstudie (Daten)

Collected Data	Release 1	Release 2	Release 3	Release 4	Release 5	Correction Release	Total
Calendar time (weeks)	2	2	2	1	1	0.4	8.4
# User stories implemented	5	9	9	4	3	4	34
# Tasks defined	10	30	18	21	19	9	107
Task effort (median, h)	11.7	2.9	5.9	1.7	2.6	0.7	2.7
Post-release defects/KLoc	2.19	2.10	2.04	8.70	13.06	-	1.43 (3.75)
Required customer involvment (%)	17.4	21.4	18.6	25.0	23.4	24.3	20.6

	suggestions made by testers						
19	Pair programming (%)	81.7	76.3	73.0	78.8	54.2	90.4	75.9
20	Required customer involvement (%)	17.4	21.4	18.6	25.0	23.4	24.3	20.6
21	Rework costs (%)	-	8.7*	11.8	11.6	2.6	61.5	9.8

*includes also enhancements

Table 2. Exploratory data from 5+1 releases

Ermittelte Daten

aus: [Abrahamsson2004], Seite 4

5. Zusammenfassung

Zusammenfassung

Basis agiler Methoden:

- inkrementelle Entwicklung in Zyklen
- Reflexion, Feedback, Kommunikation
- Konzentration auf das eigentliche Produkt
- Akzeptanz später Anforderungsänderungen (meistens)

Scrum:

- Management-Orientierung
- Vorgehen schnell erlernbar
- Transparenz & Empirie als zentrale Ideen

eXtreme Programming:

- Technik-Orientierung
- Best Practice Umsetzungen
- Kommunikation, unbeschränkte Anforderungsänderung, schnelle & kleine Schritte als zentrale Ideen

Eigene Bewertung

+ Gute Argumente für Agile Methoden (Anforderungsänderungen, früher Nutzen, Anwenderakzeptanz)

- fehlgeleitete Schlussfolgerungen möglich
  - Gesamtüberblick verlieren
  - Verbesserung der Methoden/Prozesse vernachlässigen
  - Anpassung an spezifisches Projekt, Team und die Organisation vernachlässigen
  - *Prozesse und Werkzeuge, (umfassende) Dokumentation, Vertragsverhandlung, das Befolgen eines Plans* ignorieren

! Kontext & Aufgabenstellung beachten
  - nicht sinnvoll bei vollspezifizierten Systemen, die wenig Interaktionen mit Nutzern erfordern

➔ sinnvoll als Leitlinie auch für unerfahrene Entwickler & Wissenschaftler (Strukturierung, Best Practices)

➔ in der Wissenschaft nicht immer anwendbar (kleine Teams, verteilte Teams, kein Kunde, Anforderung bekannt)

➔ gute Prozess-Erweiterbarkeit (Datenerhebung)

Literatur

Gründe für Frameworks & Vorgehen vor agiler Softwareentwicklung angelehnt an:

- http://agilemanifesto.org/history.html
- H. Benington; 1956; Production of Large Computer Programs
- http://en.wikipedia.org/w/index.php?title=Agile_software_development&oldid=663229066#Predecessors

Schwierigkeiten der Softwareentwicklung; Entstehung von Scrum basierend auf:

- http://www.scrumguides.org/history.html

Grundlagen & Ideen, Prinzipien von agiler Softwareentwicklung basierend auf:

- http://agilemanifesto.org/history.html
- http://agilemanifesto.org/
- http://agilemanifesto.org/principles.html

Literatur

Eigenschaften, Voraussetzungen agiler Softwareentwicklung basierend auf:

- http://agilemanifesto.org/ und Gemeinsamkeiten von Scrum und eXtreme Programming

Scrum:

- K. Schwaber, J. Sutherland; 2013; Der Scrum Guide – Der gültige Leitfaden für Scrum: Die Spielregeln (deutsche Übersetzung)

eXtreme Programming Idee:

- [Beck2005]

eXtreme Programming Eigenschaften/Praktiken:

- [Beck2005]
- http://de.wikipedia.org/w/index.php?title=Extreme_Programming&oldid=142876153

eXtreme Programming Rollen, Projektablauf:

- http://de.wikipedia.org/w/index.php?title=Extreme_Programming&oldid=142876153
- http://www.it-agile.de/wissen/methoden/extreme-programming/

Literatur

Zitate, Verweise, Belege

[1] 2010; **Studie: Agile Softwareentwicklung ist Mainstream** http://heise.de/-912207

[2] A. Komus; 2014; **Status Quo Agile 2014** http://www.hs-koblenz.de/rmc/fachbereiche/wirtschaft/forschung-projekte/forschungsprojekte/status-quo-agile/

[Beck2005] K. Beck; 2005; **Extreme Programming Explained: Embrace Change – 2. Edition**

[wiki1] http://de.wikipedia.org/wiki/Extreme_Programming?oldid=140537067#Aufbauorganisation

[Rumpe2002] B. Rumpe, A. Schröder; 2002; **Quantitative Survey on Extreme Programming Projects** (http://arxiv.org/pdf/1409.6599)

[Abrahamsson2004] P. Abrahamsson, J. Koskela; 2004; **Extreme Programming: A Survey of Empirical Data from a Controlled Case Study** (http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=1334895)