Agiles Projektmanagement

Agile Scrum Foundation

Version 2.1.1 – Deutsch

Inhalt

11. Weitere agile Methoden und Verfahren

Überblick

Zwecke

- Reduzierung von Planungs- und Entwurfsaufwand
- Möglichst frühe Fertigstellung funktionierender Produkte durch iteratives Vorgehen
- Regelmäßige enge Abstimmung mit Kunden über die Produktanforderungen
- Erhöhung der Kundenzufriedenheit durch flexibles Eingehen auf Kundenwünsche

Agile Frameworks / Methoden / Prozesse / Verfahren

(Beispiele)

- Extreme Programming (XP)
- Crystal Family
  (abhängig von Personen und Risiken),
  z. B. Crystal Clear, Crystal Red,
  Crystal Blue, ...
- Kanban
- Dynamic Systems Development Method (DSDM),
  z. B. MoSCoW-Priorisierung
- Feature Driven Development (FDD),
  z. B. Plan by Feature

Agile Methoden und Entwicklungstechniken

Agile Methoden / Entwicklungstechniken

- dienen dazu, den Entwicklungs-aufwand möglichst gering zu halten.
- sind Entwicklungsmethoden, die an Agilität ausgerichtet sind.
- Grundsatz für die Anwendung agiler Methoden ist die Kritik an plan-mäßiger (Software)Herstellung:
  - *„Je mehr Du nach einem Plan arbeitest, umso mehr bekommst Du, was Du geplant hast und nicht das, was Du eigentlich benötigst.“*

Beispiele

- Pair Programming
- Story Cards
- Test Driven Development
- Refactoring
- Hintergrundkommunikation (osmotic communication)
- ...

edu bs

Agile Scrum Foundation

12. eXtreme Programming (XP)

Überblick

- eXtreme Programming (XP) umfasst verschiedene Einzelmethoden und -techniken, die iterativ angewendet werden.
- Im Mittelpunkt stehen dabei folgende Werte:
  - Respekt
  - Mut
  - Einfachheit
  - Feedback
  - Regelmäßige Kommunikation zwischen allen Beteiligten

Annahmen, auf denen XP basiert

- Der Kunde kennt selbst noch nicht vollständig alle Anforderungen an die zu entwickelnde Software.
- Das Entwicklerteam verfügt nicht über alle für Aufwandsschätzungen benötigten Informationen.
- Während des Projekts verändern sich Anforderungen und Prioritäten.

Prinzipien

- Menschlichkeit
- Wirtschaftlichkeit
- Beidseitiger Vorteil
- Selbstgleichheit (Übertragbarkeit)
- Verbesserungen
- Vielfältigkeit
- Reflexion
- Ständige Lauffähigkeit
- Gelegenheiten wahrnehmen
- Redundanzen vermeiden
- Fehlschläge hinnehmen
- Qualität
- Kleine Schritte
- Akzeptierte Verantwortung

Techniken und Praktiken – Überblick

1. Paar-Programmierung (pair programming / pairing)
2. Aufgabenzuweisung (assignment) / kollektives Eigentum (code sharing)
3. Einfaches Design (simple design)
4. Testgetriebene Entwicklung (test driven development - TDD)
5. Standardisierter Code (code standards)
6. Refactoring
7. Laufende Integration (continuous integration)
8. Go Home! (constant pace)
9. Daily Standup
10. Tracking
11. Risikomanagement (risk management)
12. Spiking

1. Paar-Programmierung (pair programming / pairing)

- Zwei Programmierer arbeiten gemeinsam an einem Computer.
- Es gibt zwei Rollen, die regelmäßig getauscht werden
  - Codierer (driver) – der Autor des Softwarecodes
  - Partner (observer) – hat den Überblick und das Ziel vor Augen, berät und kontrolliert den Codierer.
- Laufender Wissensaustausch steigert die Fähigkeiten der Entwickler und verbessert die Teamintegration (team building).
- Führt zu Qualitätssteigerung, Designverbesserung und schneller Fehlerfindung (Vier-Augen-Prinzip)
- Ermöglicht personelle Stellvertretung („bus factor“)

www.cantabriatic.com/aprendiendo-tecnicas-de-desarrollo-agil/

2. Aufgabenzuweisung / kollektives Eigentum

- Es gibt keine Aufgabenzuweisung;
  ein Entwickler / Entwicklerpaar wählt sich bei Bedarf selbständig eine Aufgabe zur Bearbeitung („pull“ anstatt „push“).
- Ein Entwickler / Entwicklerpaar kann zwar eine Aufgabe bearbeiten; die Verantwortung dafür liegt jedoch im gesamten Team und somit bei jedem einzelnen Entwickler (collective code ownership).
- Es gibt kein Monopol von Einzelwissen und keine Einzelerfolge; nur das Team kann erfolgreich sein.
- Wissensaustausch wird durch regelmäßige, geeignete Maßnahmen, z. B. pair programming erreicht.

- Auch „code sharing“ genannt.

3. Einfaches Design (simple design)

- Möglichst einfaches Design um genau das Gewünschte (und nicht mehr) zu erreichen.
- Einfaches Design heißt
  - alle Tests werden bestanden
  - kein doppelter Code vorhanden
  - verwendet möglichst wenige Klassen und Methoden
  - zeigt jederzeit deutlich die Absicht des Entwicklers
- Einfaches Design wird durch die Anwendung von ⇨ 6. Refactoring unterstützt.

4. Testgetriebene Entwicklung (test driven development)

- Vor der Entwicklung der eigentlichen Funktionalität werden die Tests dafür entwickelt.
- Es wird dann genau soviel Code entwickelt, wie für erfolgreiche Tests benötigt wird.
- Nach jedem Programmierschritt werden die Tests ausgeführt (grey-box-tests).
- Getesteter neuer Code wird laufend zum bisherigen Code hinzugefügt und als Ganzes erneut getestet (⇨ 7. continuous integration).
- Auch „test-first development“ genannt

Testtypen, z. B.

- Modultest:
  Test des neu entwickelten Moduls.
- Integrationstest:
  Test des neuen Moduls zusammen mit den bereits vorhandenen Modulen.
- Performancetest:
  Test auf geforderte Leistungsmerkmale.
- Akzeptanztest:
  Test auf Nutzbarkeit des Produkts und damit auf Kundenzufriedenheit.
- Regressionstest:
  Wiederholung von vorherigen Tests nach einer Modifikation.
- ...

5. Standardisierter Code (code standards)

- Entwicklungsstandards verbessern die Lesbarkeit des Codes und tragen zum besseren Verständnis im gesamten Entwicklungsteam bei.
- Nur durch Standards kann das Team eine gemeinsame Verantwortung tragen.
- Sprachliche Standards tragen zum besseren Verständnis zwischen Kunde und Entwickler bei (Satzbau, Namenskonventionen, …).

6. Refactoring

- Manuelle oder automatisierte Verbesserung von Quelltext ohne funktionale Veränderung.
- Ziel:
  Reduzierung des Aufwands für Fehleranalyse und Erweiterungen.
- Verbesserung von ...
  - Lesbarkeit
  - Verständlichkeit
  - Wartbarkeit
  - Erweiterbarkeit
  - Struktur
- Wird zur Erreichung von einfachem Design verwendet (⇨ 3. Einfaches Design).

7. Laufende Integration (continuous integration)

- Durch laufende Integration werden einzelne Komponenten regelmäßig und in kurzen Zeitabständen in ein lauffähiges Gesamtsystem (Produktinkrement) eingebunden.
- Es entsteht eine hohe Integrationsroutine bis hin zur Automatisierung.
- Dadurch können Integrationskosten reduziert werden.
- Fehler werden frühzeitig gefunden; ihre Behebung verursacht ebenfalls geringere Kosten.
- Auch „permanent Integration“ genannt.

8. Go Home! (constant pace)

- Arbeitszeit nicht mehr als 40 h / Woche
- Überstunden stören das Erreichen eines regelmäßigen Entwicklungsrhythmus und verringern …
  - die Freude an der Arbeit
  - die Konzentrationsfähigkeit der Entwickler
  - die Qualität des Produkts
- Scheinbar notwendige Überstunden beruhen auf defizitärer Planung.

9. Daily Standup

- Tägliches Meeting der Entwickler zum Austausch und Synchronisierung
- Verbessert Kommunikation und Transparenz
- Dauert 10-15 Minuten und findet im Stehen statt.
- Vergleichbar mit dem „Daily Scrum“-Meeting in Scrum.
- Auch hier können reihum die drei Fragen aus dem „Daily Scrum“-Meeting beantwortet werden
  - Was wurde seit dem letzten Daily Standup erreicht?
  - Was soll bis zum nächsten Daily Standup erreicht werden?
  - Welche Hindernisse treten dabei auf?

Rollen: Tracking / Risikomanagement

10. Tracking

- XP sieht die Rolle des Trackers vor.
- Aufgabe des Trackers ist es, täglich den Fortschritt im Projekt und die Leistung des Teams zu messen. Er wird dabei ggf. vom gesamten Team unterstützt.
- Die Rolle des Trackers sollte häufig neu zugewiesen werden.

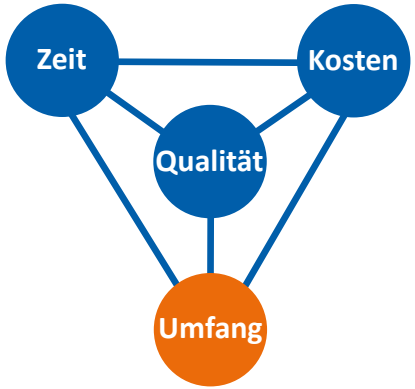
11. Risikomangement

- XP sieht die Rolle des Schwarzsehers / Schwarzmalers (doomsayer) vor.
- Aufgabe des Schwarzsehers ist es, bei jedem einzelnen Teammitglied ein Bewusstsein für Risiken und Probleme zu schaffen.
- In Meetings werden dann die Risiken und Probleme thematisiert und ein angemessener Umgang mit ihnen vereinbart.

12. Spiking

- Spiking bezeichnet das schnelle Ausprobieren (quick & dirty) eines Stücks Entwicklungscode.
- Beim Spiking werden keine der üblicherweise in XP angewendeten Techniken / Praktiken (einfaches Design, Paar Programmierung, testgetriebene Entwicklung, …) berücksichtigt.
- Der beim Spiking verwendete Code darf nicht in die Produktentwicklung übernommen werden.
- Nach Beendigung des Spikings kehrt der Entwickler wieder zu seiner normalen Entwicklungsarbeit zurück und lässt die Erkennisse des Spikings unter Beachtung der verwendeten Regeln dort einfließen.

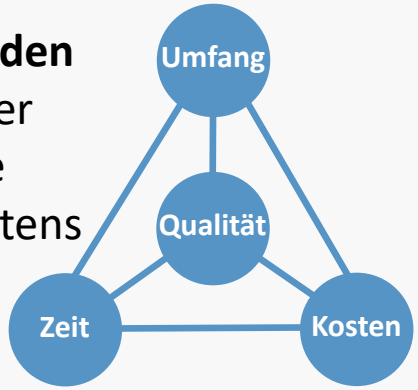
13. Dynamic Systems Development Method (DSDM)



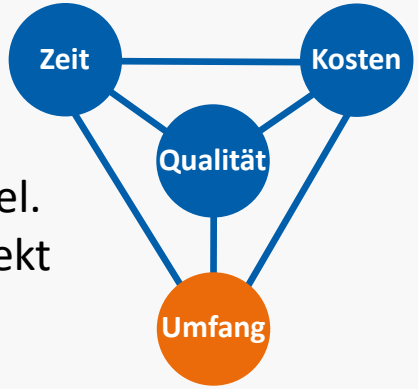
Überblick

- Agile Methode, gut geeignet auch für große Projekte und mehrere Teams
- Definiert eine Vielzahl eigener Rollen, z. B. Projektmanager
- Weitere Merkmale …
  - Die Projektzeit ist auf die Entwicklung wirklich wichtiger Dinge begrenzt.
  - Planung
  - MoSCoW-Priorisierung
  - Ausnahmen
  - Selbst-Organisation
  - Vertragsgestaltung

- **Klassische Methoden**
  Die Änderung einer Größe bedingt die Änderung mindestens einer anderen Größe.



- **DSDM**
  Zeit und Kosten sind fix, der Umfang ist variabel. Das gesamte Projekt hat eine Timebox.



Planung und Priorisierung

Planung

- Die Planung erfolgt in zwei Schritten:
  - Die Grobplanung findet vor dem Projekt statt und ermöglicht die Festlegung von Zeit und Kosten auf Basis einer Priorisierung.
  - Die Feinplanung der einzelnen Features erfolgt während der agilen Iterationen.
- Eine detaillierte Feinplanung vor oder zu Beginn des Projekts würde die Anwendung agiler Methodik unmöglich machen.

Priorisierung

- MoSCoW ist eine Technik zur Priorisierung von Features.
- Sie basiert auf Dringlichkeit und Auswirkung eines Features und stellt somit seinen Einfluss auf den Geschäftswert in den Mittelpunkt.
- Die MoSCoW-Priorisierung ist methodenunabhängig;
  sie kann auch in nicht-agilen Projekten oder Verfahren eingesetzt werden.

MoSCoW-Priorisierung

- **M** – Must have
  - Das Feature muss entwickelt werden, da das Produkt sonst nicht nutzbar sein wird.
  - Höchstens 60% aller Features sollten „Must have“-Features sein.
  - MUST: **M**inimal **U**sable **S**ubse**T**
- **S** – Should have
  - Das Feature sollte entwickelt werden, da sonst Probleme in der Produkt-nutzung auftreten werden.
  - Durch seine Berücksichtigung darf die Umsetzung eines „Must have“-Features nicht beeinträchtigt werden.
- **C** – Could have (Nice to have)
  - Das Feature kann entwickelt werden, wenn „Must have“- und „Should have“-Features erledigt sind.
  - Mindestens 20% aller Features sollten „Could have“-Features sein.
- **W** - Won‘t have
  - Das Feature ist nützlich, aber nicht dringlich. Seine Entwicklung wird deshalb ggf. auf einen späteren Zeitpunkt verschoben.

Ausnahme und Selbst-Organisation

Ausnahme (Exception)

- Fortschrittsmessung erfolgt durch regelmäßigen Vergleich von Erreichtem (IST) mit Geplantem (SOLL).
- Es wird der Umfang geplant, da Zeit und Kosten fix sind.
- Sollten nicht alle „Must have“-Features entwickelt werden können (Ausnahme), wird zur Entscheidung an das übergeordnete Management eskaliert.

Selbst-Organisation

- Entscheidungsbefugnisse von agilen Teams sind immer begrenzt, auch in „selbst organisierten“ Teams.
- In DSDM entscheidet das übergeordnete Management über die Entwicklung von „Must have“- und „Should have“-, das Team über die Entwicklung von „Could have“-Features.

Vertragsgestaltung

- Fixe Zeit/Kosten und variabler Umfang wirken sich auf die Vertragsgestaltung aus.
- Kunden möchten meistens (möglichst) genau wissen, welche Features sie zu welchen Kosten bekommen.
  Dies ist mit einem agilen Ansatz nicht vereinbar.
- Generell am besten geeignet ist der Vertragstyp „Zeit und Material“.
  Im Vertrag werden dann feste Kosten für benötigte Ressourcen vorgesehen
  - z. B. Personentage
  - verwendetes Material
- Variabler Umfang in der Vertragsgestaltung kann durch verschiedene Ansätze erreicht werden.
  Häufig werden konkrete Features vorab vereinbart. Ändern sich im Projekt die Anforderungen, werden neue, wichtigere Features gegen bereits vereinbarte, unwichtigere ausgetauscht.
- **Aus dem Agilen Manifest**
  Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsgestaltung!

edu bs

Agile Scrum Foundation

14. Kanban, ScrumBan und ScrumBut

Kanban

Merkmale

- Visualisierung von Tätigkeiten
  - Führt zu mehr Transparenz und somit
  - zu besserer Zusammenarbeit und
  - zu besserem Feedback
- Limited „Work in Progress“ (WiP):
  Die Anzahl gleichzeitig ausgeführter Tätigkeiten ist begrenzt.
  - Unterstützt die Fertigstellung anstatt die Entwicklung
  - Führt zu höherer Produktivität
- Pull vs. Push:
  Entwickler nehmen sich neue Aufgaben, wenn sie freie Kapazitäten haben. Es gibt keine Zuweisung von Aufgaben.
  - Führt zu sich selbst regelnder Unterstützung im Entwicklungsteam
  - Untätige Entwickler helfen dort mit, wo sich die Arbeit staut
  - Jeder Entwickler übernimmt direkt Verantwortung für die Fertigstellung des Produkts anstatt sich nur auf „seine“ Aufgaben zu konzentrieren.
- Kanban-Techniken werden häufig in anderen agilen Methoden eingesetzt.

ScrumBan

- Scrum und Kanban werden häufig miteinander kombiniert.
- Wird eine noch größere Flexibilität von Scrum benötigt, können sogar Sprints weggelassen werden. Dies wird als ScrumBan bezeichnet.

Artefakte in ScrumBan

- Das Product Backlog wird als ToDo-Spalte des Kanban-Boards dargestellt.
- Sprint Backlogs entfallen.
- Produktinkremente und die Definition of „Done“ werden unverändert verwendet.

Meetings in ScrumBan

- Sprint Planning Meetings entfallen.
- Review Meetings sind weiter erforderlich und werden nach festgelegten Intervallen (bevorzugt) oder nach Fertigstellung einer bestimmten Entwicklungsmenge abgehalten.
- Retrospektiven sind erforderlich und finden im Anschluss an Review Meetings statt.
- Daily Standups sind erforderlich und finden weiterhin täglich statt.

ScrumBut

- Um die Funktionsweise von Scrum nicht zu gefährden, dürfen seine Elemente nicht verändert, z. B. weggelassen werden.
- Unter Missachtung dieser Erkenntnis wird Scrum gelegentlich verändert, z. B.
  - We use Scrum, but we do not keep the Sprints timeboxed.
  - We use Scrum, but we set the duration of the Sprint in the Sprint Planning.
  - We use Scrum, but we do not let the Product Backlog evolve.
  - We use Scrum, but we do not find it necessary to have Sprint Retrospectives.
- Eine solche Vorgehensweise ist nicht Scrum! Sie wird als ScrumBut bezeichnet.
- Wer Scrum nur zu 95% anwendet, erhält nicht 95% seines Nutzens, sondern lediglich ca. 10-20%!

edu bs

Agile Scrum Foundation

15. Abschluss

Vom Anfänger zum Profi (1)

Schritte zur kontinuierlichen Verbesserung

- Transparenz, Inspektion und Adaption – Scrum und seine Anwendung sind fortwährend zu prüfen und zu verbessern.
- Teamentwicklung:
  Forming, Storming, Norming, Performing
  (Tuckman Modell)
- Wichtigstes Element: Iteration
- Wichtigstes Meeting: Sprint Retrospektive

Vom Anfänger zum Profi (2)

1. **Lernen – üben – anwenden**
   - Scrum lernen, ausführen und durch permanente Anwendung der Regeln verinnerlichen
2. **Erfahrungen sammeln – ausprobieren – verbessern**
   - Scrum variieren, Experimente machen, sich weiter entwickeln und verbessern
3. **Neue Wege gehen – vollenden**
   - Perfektion in der Zielerreichung – unabhängig von vorgeschriebenen Vorgehensweisen

Lehrling

gehorchen, befolgen

Geselle

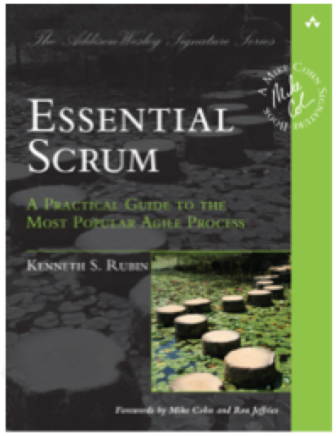
durchbrechen, variieren

Meister

sich trennen, entfernen

Copyright / Lizenz

- Slides in this presentation contain items from the Visual AGILExicon®, which is a trademark of Innolution, LLC and Kenneth S. Rubin.
- The Visual AGILExicon is used and described in the book: ***Essential Scrum: A Practical Guide to the Most Popular Agile Process***.
- You can learn more about the Visual AGILExicon and permitted uses at: http://innolution.com/resources/val-home-page



Visual AGILExicon®

16. Anhang


EXIN
EXIN
Agile Scrum
FOUNDATION
Certified by
EXIN
Musterprüfung
Ausgabe 202001