2.6 Agile Softwareentwicklung

Agile Softwareentwicklung

- Agile Softwareentwicklung ist der Oberbegriff für den **Einsatz von Agilität** (lat. agilis 'flink, beweglich') in der Softwareentwicklung.
- Agile Softwareentwicklung zeichnet sich durch **geringen bürokratischen Aufwand** und wenige, flexible Regeln aus.
- Beispiel: eXtreme Programming

Agile Softwareentwicklung Zielsetzung

- Das Ziel Agiler Softwareentwicklung ist es, den **Softwareentwicklungsprozess flexibler und schlanker** zu machen, als das bei den klassischen Vorgehensmodellen der Fall ist.
- **Gegenbewegung** gegenüber oft als schwergewichtig und bürokratisch angesehenen Softwareentwicklungsprozessen
- Versuch, einen **Kompromiss zwischen keinem Prozess und zu viel Prozess** zu finden, so dass gerade soviel Prozess vorhanden ist, damit sich der Aufwand lohnt.
- Es werden **möglichst wenig Dokumente** gefordert, im Extremfall ist der Code das Dokument.
- Iterative Entwicklung mit **häufig ausgelieferten lauffähigen Versionen** des Zielsystems, die jeweils eine Teilmenge der geforderten Eigenschaften enthalten. Die lauffähigen Versionen sollen voll integriert und so sorgfältig getestet werden wie eine endgültige Auslieferung.
- **Stabile Pläne sind Pläne für kurze Zeiträume** und werden für jeweils eine einzelne Iteration gemacht.

Geschichte

- Ansätze Anfang der 1990er Jahre
- 1999, Kent Beck und andere veröffentlichten das erste Buch zu Extreme Programming
- Ende 2005: 14% der Unternehmungen in Nordamerika und Europa ihre Software unter Zuhilfenahme von agilen Prozessen entwickeln. Weitere 19% denken über die Nutzung nach.

- Umfrage [Ambler06] unter 4000 Personen Einsatz folgender agiler Modelle in absteigender Reihenfolge
  - X2: eXtreme Programming
  - FDD - Feature Driven Development
  - Scrum
  - AUP (Agile Unified Process)
  - Agile MSF (Microsoft Solutions Framework)
  - Crystal-Familie
  - DSDM (Dynamic System Development Method)

Managementbereiche

Managementbereiche	*XP*	*Scrum*	*Crystal*
Technik	++	o	+
Auftraggeber	o	o	o
Anwender	+	+	+
Team	–	++	++
Organisation	o	o	o

Abdeckungsgrad der Managementbereiche durch agile Modelle nach [Bal08]
Einsatz:

- XP, wenn weniger als 12 EntwicklerInnen mitarbeiten und ein automatischer Integrationstest möglich ist.
- Crystal, wenn das Team zu groß ist und möglichst wenig Disziplin erforderlich sein soll. Crystal Clear ist die leichteste der agilen Modelle

---

2.6.1 Manifest agiler Software-Entwicklung

Manifest agiler Software-Entwicklung

- Februar 2001 “Agiles Manifest“ (englisch Manifesto for Agile Software Development oder kurz **Agile Manifesto**)
- `http://www.agilealliance.org/`

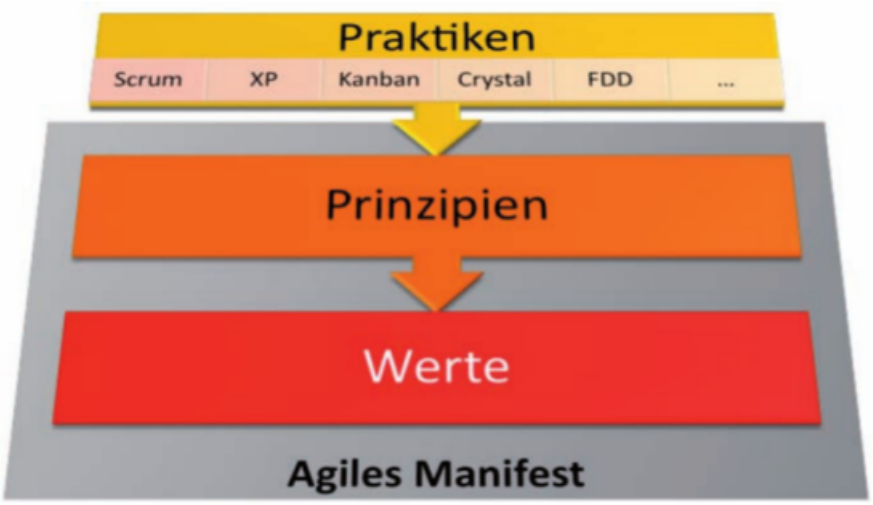
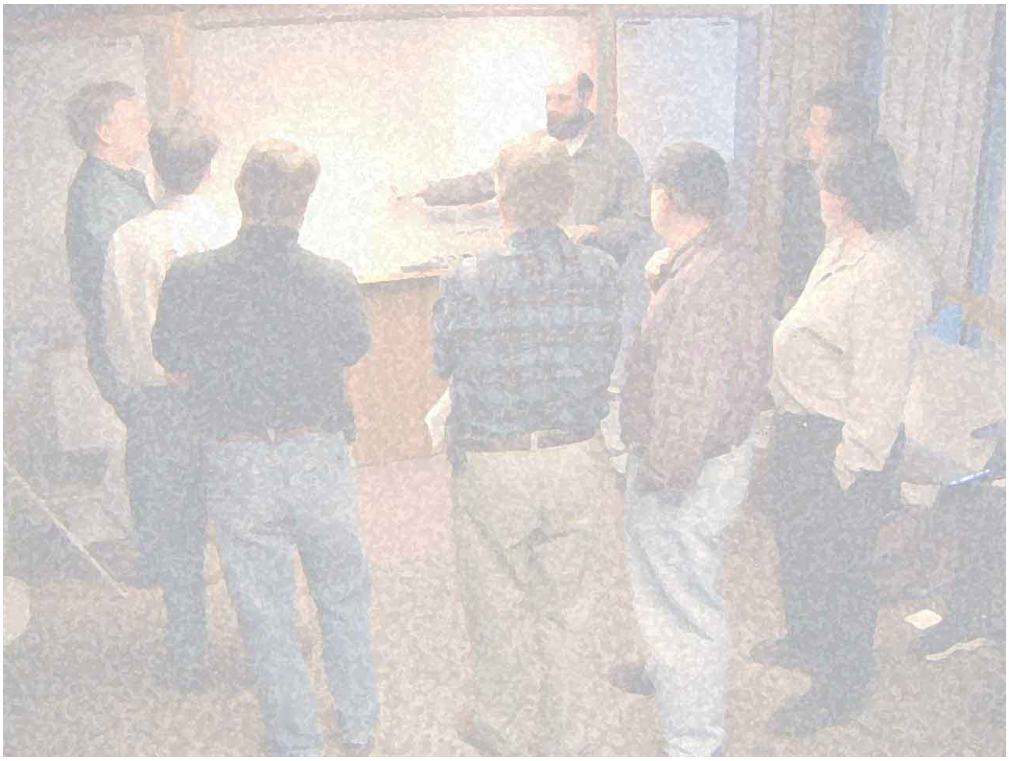


Bild 2.41: Werte, Prinzipien und Praktiken, aus [Col11]



Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

---

1. **Individuen und Interaktionen gelten mehr als Prozesse und Werkzeuge.**
   - Zwar sind wohldefinierte Entwicklungsprozesse und hochentwickelte Entwicklungswerkzeuge wichtig, wesentlich wichtiger ist jedoch die Qualifikation der Mitarbeitenden und eine effiziente Kommunikation zwischen ihnen.
2. **Funktionierende Programme gelten mehr als ausführliche Dokumentation.**
   - Gut geschriebene Dokumentation kann zwar hilfreich sein, das eigentliche Ziel der Entwicklung ist jedoch die fertige Software. Außerdem ist eine intuitiv verständliche Software hilfreicher als umfangreiche Dokumentation, die sowieso niemand liest.
3. **Die stetige Zusammenarbeit mit dem Kunden steht über Verträgen.**
   - Ein Vertrag ist normalerweise die Grundlage für die Zusammenarbeit. Was der Kunde aber wirklich will, kann nur in ständiger Kommunikation mit ihm ermittelt werden.
4. **Der Mut und die Offenheit für Änderungen steht über dem Befolgen eines festgelegten Plans.**
   - Im Verlauf eines Entwicklungsprojektes ändern sich viele Anforderungen und Randbedingungen ebenso wie das Verständnis des Problemfeldes. Das Team muss darauf schnell reagieren können.

Prinzipien: Die agile Allianz

Wir befolgen diese Prinzipien:

- Unsere höchste Priorität liegt darauf, den Kunden durch frühzeitige und kontinuierliche Auslieferung einsetzbarer Software zufriedenzustellen.
- Wir begrüßen sich ändernde Anforderungen, auch in einem späten Entwicklungsstadium. Agile Prozesse werden geändert, damit der Kunde einen Wettbewerbsvorteil hat.
- Wir liefern häufig lauffähige Software in einer Zeitspanne von mehreren Wochen und mehreren Monaten aus, wobei wir kürzere Zeiten bevorzugen.
- Management und Entwickler müssen täglich im Projekt zusammenarbeiten.
- Projekte basieren auf motivierten Mitarbeitern. Wir geben ihnen die Umgebung und die Unterstützung, die sie benötigen, und trauen ihnen zu ihren Job zu tun.
- Die effizienteste und effektivste Methode, um Informationen in einem Entwicklungsteam zu vermitteln, ist von Angesicht zu Angesicht.

- Lauffähige Software ist das primäre Fortschrittsmaß.
- Agile Prozesse unterstützen eine nachhaltige Entwicklung. Auftraggeber, Entwickler und Benutzer sollen in der Lage sein, die Entwicklung permanent mit zu vollziehen.
- Die ständige Beachtung technischer Perfektion und guter Entwürfe unterstützt die Flexibilität.
- Einfachheit - die Kunst nicht notwendige Arbeit nicht zu tun - ist essenziell.
- Die besten Architekturen, Anforderungen und Entwürfe stammen von sich selbst organisierenden Teams.
- In regelmäßigen Abständen reflektiert das Team darüber, wie es effektiver werden kann und passt das eigene Verhalten dann entsprechend an.

2.7 XP: eXtreme Programming

Geschichte

- Ansätze Anfang der 1990er Jahre
- 1999, Kent Beck und andere veröffentlichten das erste Buch zu Extreme Programming
- kam einer Revolution gleich
- 1997 wurde V-Modell 97 veröffentlicht
- XP fordert Verzicht auf Dokumente
- XP wurde im Wesentlichen von Kent Beck, Ward Cunningham und Ron Jeffries entwickelt.

XP: Werte

- **Einfachheit** (simplicity): Ziel ist es, möglichst einfache Lösungen zu finden, da sie schneller und preiswerter zu entwickeln und zu warten sind (The simplest thing that could possibly work).
- **Kommunikation** (communication): Durch intensive - am besten persönliche - Kommunikation der Beteiligten werden Informationen am effektivsten ausgetauscht, so dass auf einen Großteil der sonst üblichen, schriftlichen Dokumentation verzichtet werden kann.
- **Rückkopplung** (feedback): Qualität wird durch Rückkopplung auf verschiedenen Ebenen sichergestellt. Entwickler schreiben Komponententests, um eine Rückkopplung über die Fehlerfreiheit ihrer Programme zu erhalten. Anwender arbeiten mit neuen Versionen, um zu prüfen, ob die Software die gewünschte Aufgabe erledigt.
- **Mut** (courage): Um die oben aufgeführten Werte zu erfüllen, ist Mut erforderlich. Einfachheit erfordert den Mut zur Lücke, den Mut etwas wegzulassen. Kommunikation verlangt die Auseinandersetzung mit anderen Meinungen. Rückkopplung erfordert Mut, mit negativen Resultaten umzugehen.

XP: Prinzipien

Aus den XP-Werten leiten sich 15 XP-Prinzipien ab. Die fünf wichtigsten Prinzipien sind:

- **Unmittelbare Rückkopplung** (rapid feedback): Rückkopplungen sollen so schnell und so früh wie möglich erfolgen, da dann der Lernerfolg am größten ist.
- **Einfachheit anstreben** (assume simplicity): Einfache Lösungen sind schneller zu entwickeln, besser zu verstehen. leichter zu ändern usw.
- **Inkrementelle Änderungen** (incremental change): Notwendige Änderungen in kleinen Schritten vornehmen, um die Risiken zu reduzieren.
- **Neues wollen** (embracing chance): Fortschritt kann nur durch Neues, durch verändern der Gegenwart erreicht werden. Stabile Verhältnisse müssen verlassen werden, Risiken müssen eingegangen werden, um Neues zu etablieren.
- **Qualitätsarbeit** (quality work): Jede/r Beteiligte hat Qualitätsarbeit zu liefern, wobei die Qualitätsmaßstäbe klar sein müssen.

XP: weitere 10 Prinzipien

- **Lernen lehren** (teach learning): Die Beteiligten sollen selbst lernen, was sie benötigen und wie sie vorgehen, um die Ziele zu erreichen.

- **Geringe Anfangsinvestition** (small initial investment): Mit geringem Budget beginnen, um sich auf die wichtigen Aspekte konzentrieren zu müssen und um bei Abbruch das finanzielle Risiko klein zu halten.

- **Auf Sieg spielen** (play to win): Projekte werden begonnen, um sie erfolgreich zu beenden. Umgekehrt werden Projekte beendet, wenn sie nicht "gewonnen"werden können.

- **Gezielte Experimente** (concrete experiments): Durch gezielte Experimente soll geprüft werden, ob Entwicklungsentscheidungen oder Prozessabläufe richtig oder falsch sind.

- **Offene, ehrliche Kommunikation** (open, honest communication): Das Verschweigen oder Verbergen von Problemen und Entwicklungsrückständen ist für ein Projekt tödlich. Das Herstellen einer offenen und ehrlichen Kommunikation ist daher eine zentrale Aufgabe für das Softwaremanagement.

- **Team-Instinkte nutzen, nicht dagegen arbeiten** (work with people's instincts, not against them): Wenn das ganze Team etwas für gut oder schlecht hält, dann sollte man dem Team-Instinkt folgen.

- **Verantwortung übernehmen** (accepted responsibility): Das Team und jeder einzelne sollen bereit sein, Verantwortung für eine Aufgabe zu übernehmen.

- **An lokale Rahmenbedingungen anpassen** (local adaptations): XP sollte an die lokalen Rahmenbedingungen angepasst werden.

- **Mit leichtem Gepäck reisen** (travel light): Das "Marschgepäck"für die Projektdurchführung sollte sorgfältig ausgewählt und flexibel kombinierbar sein.

- **Ehrliches Messen** (honest measurement): Die Messungen im Laufe eines Projekts müssen vom Team ehrlich durchgeführt werden, damit nicht ein falscher Projektfortschritt vorgetäuscht wird.

XP: Praktiken: Rollen

Rolle	Beispiel	Aufgaben
Product Owner	Produktmanagement, Marketing, ein User, Kunde, Manager des Users, Analyst, Sponsor	Hat Verantwortung, setzt Prioritäten, Entscheider für bestes ROI (Return on Investment)
Kunde	Auftraggeber, kann auch der Product Owner sein, kann, muss aber nicht User sein	Entscheidet, was gemacht wird, gibt regelmäßig Feedback, Auftraggeber
Developer	Bestandteil des Teams, das ganze Entwicklungsteam besteht aus Entwicklern: Programmierer, Tester, DB-Experten, Architekt, Designer	Entwickelt das Produkt
Projektmanager	Ist gewöhnlich der Product Owner. Kann auch Entwickler aber nicht Manager des Teams sein	Führung des Teams
User	Der Nutzer des zu erstellenden Produktes	Wird das zu erstellende Produkt nutzen

XP: Praktiken

Managementpraktiken	*Teampraktiken*	*Programmierpraktiken*
Kunde vor Ort	Metapher	Testen
Planungsspiel	Gemeinsame Verantwortlichkeit	Einfacher Entwurf
Besprechungen im Stehen	Fortlaufende Integration	Re-Strukturierung
Kurze Releasezyklen	Programmierstandards	Paarweises Programmieren
Retrospektiven	Nachhaltiges Tempo	

- Kunde vor Ort (on-site. customer): Kunden und Anwender (wobei XP keinen Unterschied zwischen beiden Gruppen vornimmt) stehen dem Entwicklungsteam als Ansprechpartner für fachliche Fragen in der Regel permanent zur Verfügung und sind Teil des Teams. Sie schreiben kein Pflichtenheft oder ähnliches. Im Rahmen der Qualitätssicherung übernehmen Akzeptanztests im übertragenen Sinne die Rolle der Anforderungsspezifikation.
- Planungsspiel (planning game): XP-Projekte sind iterativ und inkrementell. Im sogenannten Planungsspiel wird der Umfang des jeweils nächsten Inkrements zwischen Kunden. Anwendern und Entwicklern festgelegt. Anwender und Kunden geben die Prioritäten der zu realisierenden Anforderungen an, die Entwickler schätzen die Aufwände.
- Besprechungen im Stehen (standup-meetings): Alle Projektbeteiligten treffen sich jeden Tag zu einer festgelegten Zeit zu einer Besprechung im Ste-

hen, um sich für 15 Minuten über den Projektfortschritt auszutauschen.

- Kurze Releasezyklen (short releases): Neue und geänderte Teilprodukte werden den Anwendern in kurzen Abständen zur Verfügung gestellt. Die ersten Erfahrungen der Anwender werden bei der Weiterentwicklung berücksichtigt.
- Retrospektiven (retrospectives): In der Regel soll in Abständen von einem Monat bis zu sechs Monaten eine halbtägige bis dreitägige Rückschau auf den Entwicklungsprozess vorgenommen werden, um Probleme zu identifizieren und für die weitere Entwicklung zu vermeiden.
- Metapher (metaphor): Den Entwicklern werden wenige, aber klare Richtlinien für die Entwicklung vorgegeben, die eine konsistente Produktentwicklung unterstützen.
- Gemeinsame Verantwortung (collective ownership): Alle Entwickler sind für das Produkt verantwortlich. Sämtlicher Quellcode darf von jedem Entwickler zu jedem Zeitpunkt geändert werden. Fehlt ein Entwickler, dann kann die Arbeit von einem anderen fortgesetzt werden.
- Fortlaufende Integration (continuous integration): Erweiterungen und Änderungen werden kontinuierlich in das Produkt integriert, so dass die Änderungen vorn ganzen Team getestet werden können.
- Programmierstandards (coding standards): Es werden pragmatische Programmierstandards festgelegt, damit der Quellcode einheitlich gestaltet ist.
- Nachhaltiges Tempo (sustainable pac); Um die Kreativität und Konzentration der Entwickler über eine langen Zeitraum aufrecht zu erhalten, werden Überstunden nur in engen Grenzen geduldet. Permanente Überstunden deuten auf ein Entwicklungsproblem hin.
- Testen (testing): Alle Programme müssen getestet werden. Modultests werden programmiert, die automatisch ausgeführt werden können. Akzeptanztests überprüfen die fachlichen Anforderungen. Der endgültige Akzeptanztest ist der Einsatz beim Anwender.
- Einfacher Entwurf (simple design): Das zu entwickelnde Produkt soll möglichst einfach gestaltet werden, da einfache Entwürfe schneller realisiert werden können und leichter zu verstehen sind.
- Re-Strukturierung (refactoring): Durch Änderungen und Erweiterungen entstehen "Verwerfungenïn der Produktarchitektur, die die Weiterentwicklung behindern. Durch Umstrukturierungen der Architektur unter Beibehaltung der Funktionalität werden die Defizite behoben. Durch automatisierte Tests wird geprüft, dass keine unerwünschten Seiteneffekte auftreten.
- Paarweise Programmierung (pair programming): Es darf grundsätzlich nur zu zweit programmiert werden. Jeweils zwei Entwickler sitzen an einem Computersystem. Dadurch soll die Qualität der Software erhöht werden. Durch wechselnde Paare soll sich das Wissen Ober das Produkt schnell im Team verbreiten. Da Inspektionen ein bewährtes Mittel der Qualitätssicherung sind, wird durch das paarweise Programmieren eine Art ständige Durchsicht eingeführt.

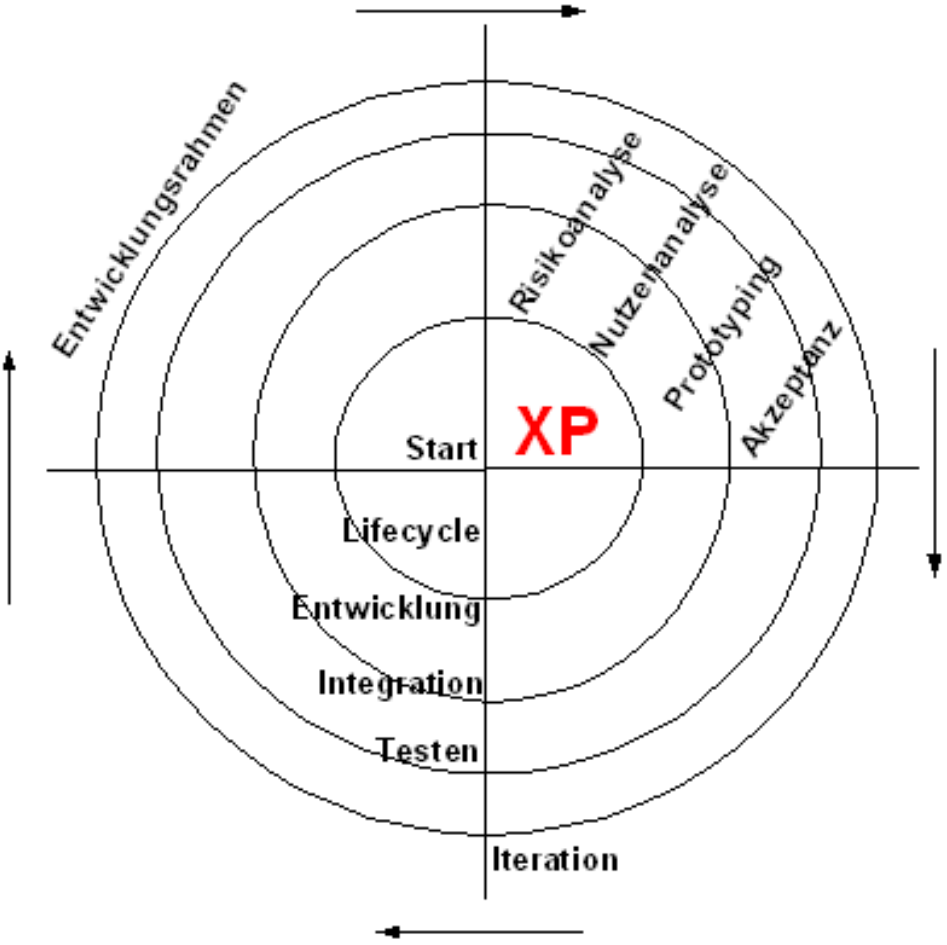


Bild 2.42: XP Lebenszyklus, aus `http://de.wikipedia.org/wiki/Extreme_Programming`

Beispiel User Story

User-Storys mit Aufwandsabschätzung in Story-Points

Story No.	Story	Abschätzung (Story Points)
1	Als Arzt kann ich alle Patienten sehen, die ich am Tage habe.	3
2	Als Arzt kann ich über die Gesundheitsgeschichte meiner Patienten Auskunft geben.	5
3	Als Assistentin kann ich einem Patienten einen Termin geben.	2
4	Als Assistentin kann ich einem Patienten eine Verschreibung ausdrucken.	1

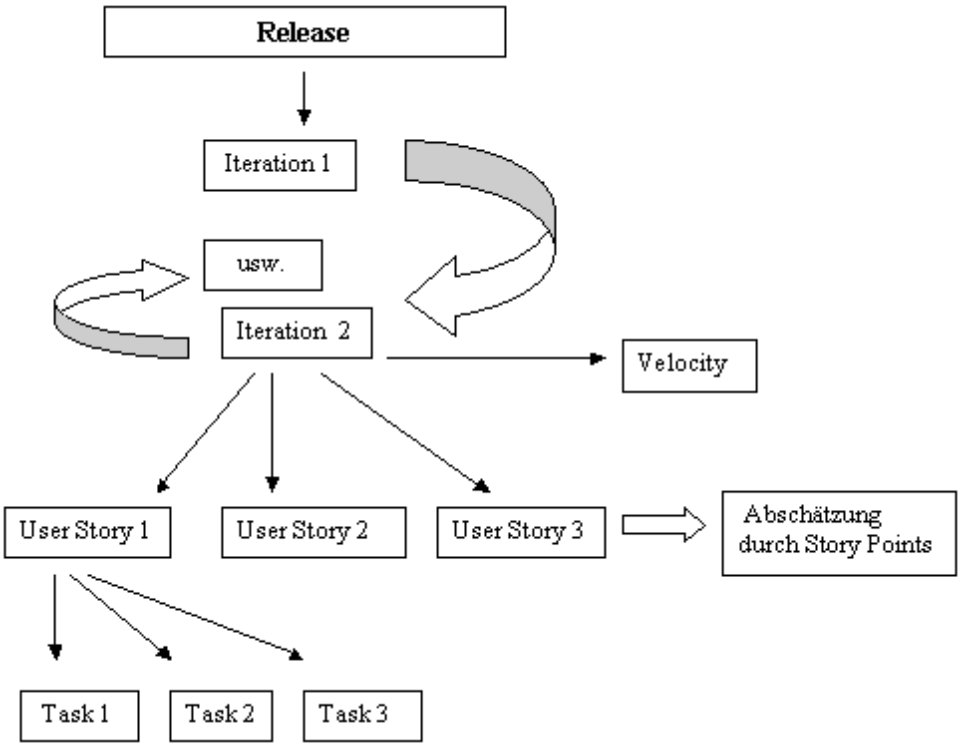


Bild 2.43: Release, Iterationen, User-Storys und Tasks, aus `http://de.wikipedia.org/wiki/Extreme_Programming`

2.7.1 Paarprogrammierung

`http://de.wikipedia.org/wiki/Paarprogrammierung`

- Bei Paarprogrammierung (auch Pair Programming genannt) handelt es sich um eine Arbeitstechnik, die sich häufig bei agilen Vorgehensweisen zur Softwareentwicklung findet.
- Paarprogrammierung ein wichtiger Bestandteil von Extreme Programming (XP)
- bei der Erstellung des Quellcodes arbeiten jeweils zwei ProgrammiererInnen an einem Rechner
- Ein/e Programmierer/in schreibt den Code, während die/der andere über die Problemstellungen nachdenkt, den geschriebenen Code kontrolliert sowie Probleme, die ihr/ihm dabei auffallen, sofort anspricht.
- Die Probleme können dann sofort (im Gespräch zu zweit) gelöst werden
- Die beiden ProgrammiererInnen sollten sich bezüglich dieser beiden Rollen abwechseln.

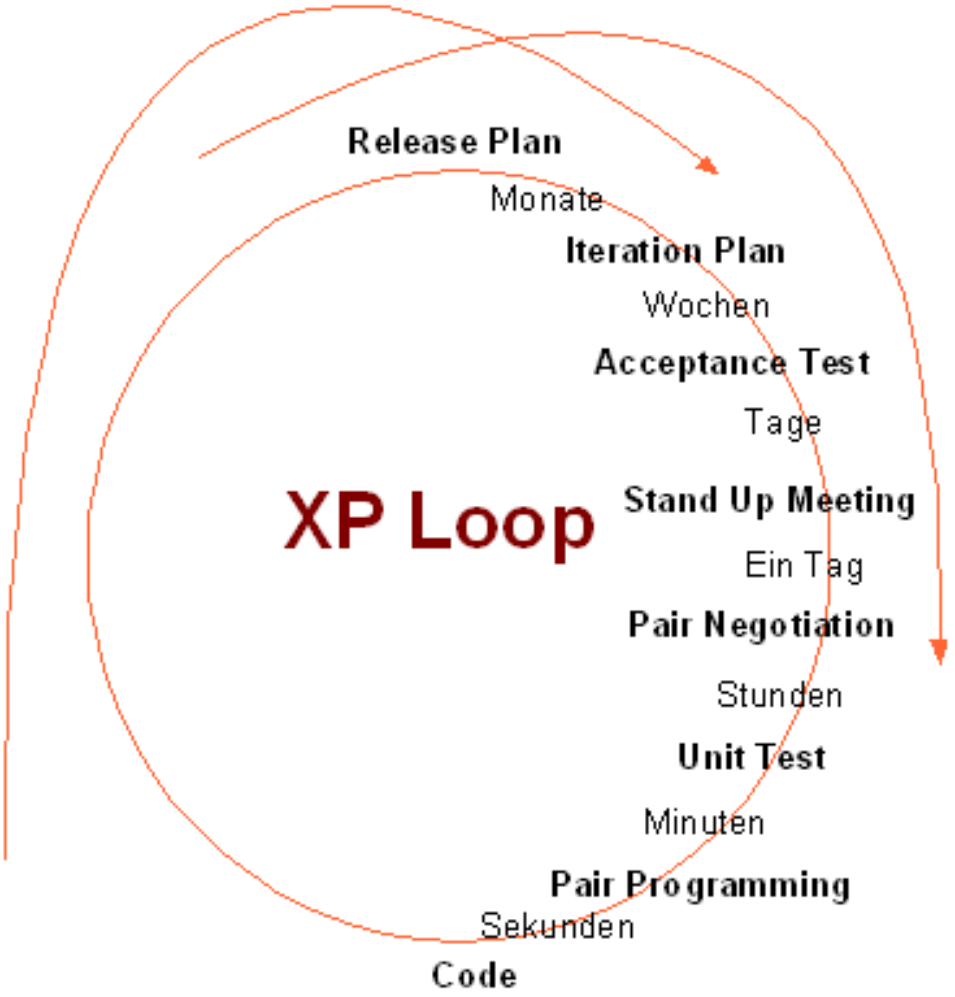


Bild 2.44: XP Kreislauf (Loop): Welche Schritte in welchen zeitlichen Abständen, aus `http://de.wikipedia.org/wiki/Extreme_Programming`

- Auch die Zusammensetzung der Paare sollte sich häufig ändern.

- Paarprogrammierung soll die Softwarequalität steigern.
- Durch die Kontrollfunktion der zweiten Person sollen problematische Lösungen vermieden werden.
- Die Paarprogrammierung dient auch zur Verbreitung von Wissen über den Quellcode.

Positive Effekte

Höhere Disziplin

- Paare entwickeln viel eher an der richtigen Stelle und machen kürzere Pausen.

Besserer Code

- Beim Pair Programming entwickelt man sich weniger leicht in Sackgassen und erreicht so eine höhere Qualität.

Belastbarerer Flow

- Pair Programming führt zwar zu einer anderen Art von Flow (lustbetontes Gefühl des völligen Aufgehens in einer Tätigkeit), ermöglicht diesen aber eher als der konventionelle Ansatz

Höhere Moral

- Pair Programming ist oft spannender und interessanter als alleine zu arbeiten.

Collective Code Ownership

- Wenn das gesamte Projektteam mit der Methode Pair Programming arbeitet und die jeweiligen Partner oft wechseln, erlangen alle Wissen über die gesamte Codebasis.

Mentoring

- Jede/r hat Wissen, das andere nicht haben. Pair Programming ist eine bequeme Möglichkeit, dieses Wissen zu verteilen.

Team building

- Die Leute lernen sich gegenseitig schneller kennen, wodurch die Zusammenarbeit verbessert werden kann.

Weniger Unterbrechungen

- Paare werden seltener unterbrochen als jemand, der alleine arbeitet.

Nachteile

Kosten

- Da sich Vorteile wie gesteigerte Qualität teils erst in späteren Phasen des Produktlebenszyklus bemerkbar machen, sind in der ursprünglichen Entwicklungsphase die **Kosten durch die doppelte Besetzung** meist höher.

Teamfindung

- **Teamfindung ist aufwendig**, nicht alle Personen können miteinander produktiv eingesetzt werden. Eingewöhnung an die Teammitglieder erfordert Zeit.

Autoritätsproblem

- Wer hat die Kompetenz, bei konträren Problemlösungen zu entscheiden, welche implementiert wird?

Zeitliche Belastun

- Wenn zusätzliche Aufgaben wie Mentoring während der Programmierung wahrgenommen werden müssen, kann es zu Verzögerungen in der Entwicklung kommen. Die TeilnehmerInnen müssen sich an unterschiedliche Programmierfähigkeiten und -stile gewöhnen.

Urheberrecht

- Es kann zu Konflikten kommen, da später nicht unbedingt klar ist, wer UrheberIn der einzelnen Passagen des Codes ist.

Haftung

- Es kann zu Konflikten kommen, da später nicht unbedingt klar ist, wer für fehlerhaften oder urheberrechtsverletzenden Code haftet.

Teamgröße

- Bei steigender Zahl von Programmierer/innen wird es schwieriger zu kommunizieren, wie Probleme zu lösen sind.
    - Deshalb ist diese Arbeitsweise **eher für kleinere Teams** geeignet.

2.7.2 Testgetriebene Entwicklung

- testgetriebene Entwicklung (auch testgesteuerte Programmierung, engl. test first development oder test-driven development, Abkürzung TDD)
- Agile Methode zur Entwicklung eines Computerprogramms, bei der die ProgrammiererInnen Software-Tests **vor** den zu testenden Komponenten entwickeln.
- Unit-Tests der testgetriebenen Entwicklung sind Grey-Box-Tests

---

Vorgehensweise

- Bei Unit-Tests werden Test-Gerüst und Unit-Gerüst parallel entwickelt.
- Die eigentliche Programmierung erfolgt in kleinen Mikroiterationen.
- Eine solche Iteration, die nur wenige Minuten dauern sollte, besteht aus drei Hauptschritten:

1. Schreibe einen kleinen Test für den nächsten zu implementierenden Funktionalitätshappen. Dieser Test sollte nicht funktionieren.

2. Erfülle den Test mit möglichst wenig Code, um schnell wieder zum "grünen Balken"(alle Tests laufen) zurückzukehren.
3. Räume den Code auf, dazu gehört die Entfernung von Duplikation, Einführung von notwendigen Abstraktionen und Umsetzen der Codekonventionen. Ziel dieses Aufräumens ist die einfache Form des Codes.

Vorteile von TDD

- Die Erfüllung der Anforderungen erhält eine triviale Metrik.
- Die Durchführung von Refaktorisierungen ist mit weniger Fehlern behaftet (im Idealfall fehlerfrei).
- Die einfache, schnelle Ausführung von Tests ermöglicht es den Programmierern, die meiste Zeit an einem korrekten System zu arbeiten.
- Die Unit-Testsuite stellt eine äusführbare Spezifikation"dar.

- Wir entwerfen einen Test, der zunächst fehlschlagen sollte.

Bild 2.45: Roter Balken, Test schlägt fehl, aus [Wes06]

Bild 2.46: Grüner Balken, Test ok, aus [Wes06]

Bewertung XP

Vorteile

+ Alle Abläufe werden auf die eigentliche Wertschöpfung bei der Softwareentwicklung ausgerichtet.
+ Alle Prinzipien und Praktiken helfen, sich flexibel auf Kundenwünsche einzustellen.

Nachteile

- XP funktioniert nur dann gut, wenn eine ganze Reihe von Randbedingungen, wie Teamgröße, Persönlichkeitsprofil und Qualifikation der Teammitglieder, zutreffen.

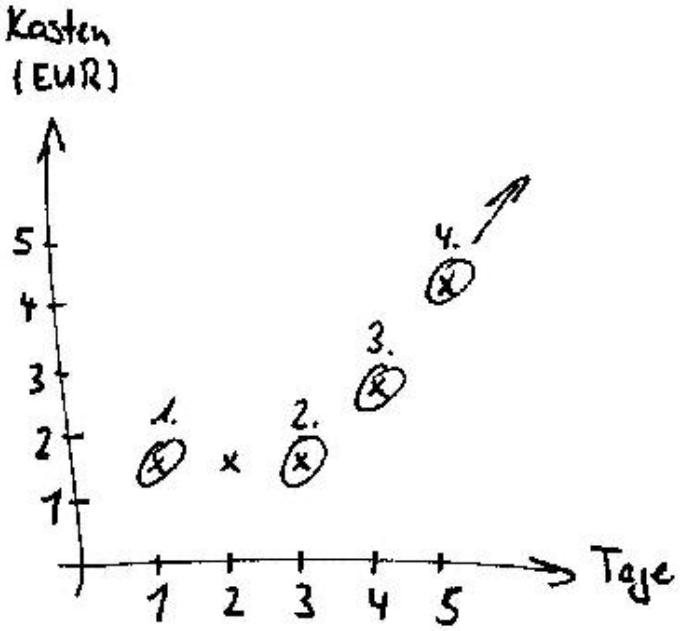


Bild 2.47: Eine neue Preiskategorie: Brainstorming einer Testroute, aus [Wes06]

- Testfälle ersetzen die Anforderungsspezifikation und auch den Entwurf. Anforderungs- und Entwurfsprobleme werden iterativ direkt beim Programmieren gelöst.
- XP verhindert, dass fachliche Problemlösungen, z. B. mit Hilfe der UML, auf einem höheren Abstraktionsniveau erfolgen - unabhängig von der konkreten technischen Umgebung und Realisierung.

XP Klassifikation

- Prozess- und Qualitätsmodell.
- Mittelgranulares Modell.
- Nicht-anpassbares Modell: Nach Auffassung von Kent Beck sind die Praktiken so eng miteinander verwoben, dass man sie entweder alle übernimmt oder es besser bleiben lässt.
- Agiles Modell.
- Softwarespezifisches Modell mit Konzentration auf das technische Management.
- Modell für die Erstellung in einem Team in einem Raum (nicht mehr als 20 EntwicklerInnen), d. h. für kleine bis mittelgroße Projekte.
- Modell für die Erstellung von Individualsoftware.

2.8 IXP Industrial XP

`http://industrialxp.org`

Werte

- **Kommunikation** (communication): analog wie XP.
- **Freude** (enjoyment): Alle Beteiligten haben Spaß am Job und bringen sich voll ein.
- **Lernen** (learning): Jede Entwicklung wird als Lernprozess verstanden. Man lernt etwas über die Anforderungen, über die Projektbeteiligten, über die Methoden und Techniken.
- **Einfachheit** (simplicity): analog wie in XP.
- **Qualität** (quality): Entwickler identifizieren sich mit Qualitätsarbeit.

Management-Praktiken

- **Bereitschaftsbeurteilung** (readiness assessment): Beurteilung, ob die Organisation und das Team für den Einsatz von IXP bereit sind.
- **Kontinuierliches Risikomanagement** (continuous risk management): Ständige Beobachtung und Bewertung von Risiken.
- **Projektskizze** (project chartering): Eine Art einfaches Pflichtenheft mit Projektziel, Kundennutzen, Erfolgskriterien, verfügbare Ressourcen, das während der Entwicklung angepasst und erweitert werden kann.
- **Projektgemeinschaft** (project community): Alle, die zum Projektteam gehören oder das Projekt beeinflussen. Wichtig, um alle richtig einzubinden.
- **Testgetriebenes Management** (test-driven management): Die Überprüfung von Projektzielen erfolgt durch sogenannte Managementtests. Solche Tests sollen **SMART** sein:
  - spezifisch (specific),
  - messbar (measurable),
  - zielführend (achievable),
  - relevant (relevant) und
  - zeitbasiert (time-based).

Organisations-Praktiken

- **Nachhaltiges Tempo** (sustainable pace): analog wie in XP.
- **Planungsspiel** (planning game): analog wie in XP.
- **Geschichten erzählen** (storytelling): Funktionale Anforderungen werden vom Kunden informell auf Karteikarten geschrieben und den Entwicklern erklärt.
- **Geschichten testen** (storytesting): Für jede Geschichte werden Akzeptanztests entworfen.
- **Häufige Releases** (frequent releases): In der Regel werden alle drei Monate neue Releases erstellt (in der Projektskizze festgelegt).
- **Kleine Teams** (small teams): Um den Kommunikationsaufwand klein zu halten, sind große Teams in kleine Sub-Teams mit festgelegten Aufgaben zu unterteilen.
- **Zusammen sitzen** (Sitting together): Das gesamte Team sitzt in einem Raum.
- **Kontinuierliches Lernen** (continuous learning): Auch während des Projekts bilden sich die Entwickler weiter.
- **Iterative Benutzbarkeit** (iterative usability): Die Gebrauchstauglichkeit des Produkts wird kontinuierlich durch Anwenderbefragungen und Softwareergonomen überprüft und verbessert.

Entwicklungs-Praktiken

- **Evolutionärer Entwurf** (evolutionary design): Es wird immer so viel entworfen, wie nötig ist. Der Entwurf wird ständig verbessert und neuen Anforderungen angepasst.
- **Re-Strukturierung** (refactoring): analog wie in .XP.
- **Testentwicklung durch Geschichten getrieben** (story test-driven development): Bevor mit der Realisierung von Anforderungen begonnen wird, werden die Akzeptanztests für die Anforderungen geschrieben. Eine Anforderung ist erfüllt, wenn der zugehörige Akzeptanztest erfolgreich verläuft.
- **Fachlich getriebener Entwurf** (domain driven design): Die fachlichen Anforderungen treiben den Entwurf, nicht die Technik.
- **Fortlaufende Integration** (continous integration): analog wie in XP.
- **Paarweises Arbeiten** (pairing): analog wie in XP jedoch bei agilen Tätigkeiten.
- **Gemeinsame Verantwortung** (collective ownership): analog wie in XP.

- **Programmierstandards** (coding standards): analog wie in XP
- **Retrospektiven** (retrospectives.): analog wie in XP

IXP: Bewertung

- Man merkt die industrielle Praxis.
- Ganz ohne ein - wenn auch rudimentäres Pflichtenheft (Projektskizze genannt) - geht es nicht.
- Die Anforderungen werden auf Karten notiert - ebenfalls die Akzeptanztests.
- siehe auch Bewertung zu XP

IXP Klassifizierung

IXP lässt sich - analog zu XP - wie folgt klassifizieren:

- Prozess- und Qualitätsmodell.
- Mittelgranulares Modell.
- Agiles Modell.
- Softwarespezifisches Modell.
- Modell für die Erstellung in einem Team in einem Raum.
- Modell für die Erstellung von Individualsoftware.

2.9 XP2 eXtreme Programming 2

- Ende 2004 2. Auflage des Buches eXtreme Programming
- wesentliche Änderungen
- deshalb auch XP2 genannt

XP2 Werte

- Einfachheit (XP)
- Kommunikation (XP)
- Rückkopplung (XP)
- Mut (XP)
- **Respekt** (respect) : Jeder Projektbeteiligte respektiert alle anderen.
- Die fünf Werte können um projektspezifische Werte ergänzt werden.

XP2 Prinzipien

- **Menschlichkeit** (humanity): Die sozialen Bedürfnisse der Projektbeteiligten müssen mit den geschäftlichen Interessen ausbalanciert werden.
- **Wirtschaftlichkeit** (economics): Softwareentwicklungen müssen wirtschaftlich sein. Projekte, die nicht wirtschaftlich sind, sind ein Misserfolg.
- **Wechselseitiger Vorteil** (mutual benefit): Für jeden Betroffenen soll eine Win-win-Situation entstehen.
- **Selbstähnlichkeit** (self similarity): Es sollte geprüft werden, ob erfolgreiche Lösungen auch in anderen Kontexten einsetzbar sind.
- **Verbesserung** (improvement): Ein kontinuierlicher Verbesserungsprozess ist notwendig. Aus nicht perfekten Lösungen lernt man viel.
- **Mannigfaltigkeit** (diversity): Die Vielfalt der Projektmitglieder wird als Chance begriffen, um alternative Lösungen zu diskutieren. Verschiedene Ansätze werden als Chance begriffen.
- **Reflexion** (reflection): Das Team denkt über seine Arbeit nach, um noch besser zu arbeiten.

- **Fluss** (flow): Das Projekt liefert einen ständigen Fluss an wertvoller Software, die immer etwas besser wird. Im Gegensatz dazu liefern Phasenmodelle umfangreiche Teilprodukte in langen Zeitabständen.
- **Gelegenheit** (opportunity): Probleme werden als Chance für Veränderungen aufgefasst.
- **Redundanz** (redundancy): Im Entwicklungsprozess soll es Redundanzen geben, aber nicht im Quellcode. Dadurch ist sichergestellt, dass ein Problem auch dann gelöst wird, wenn ein Mechanismus ausfällt. Beispielsweise sollen Programmfehler durch paarweises Programmieren, durch Komponententests und durch Akzeptanztests vermieden bzw. gefunden werden.

- **Fehlschlag** (failure): Fehlschläge lassen sich nicht vermeiden. Man muss daraus lernen. Es ist besser etwas auszuprobieren, als endlos darüber zu diskutieren.
- **Qualität** (quality): Reduzierte Qualität führt nicht zu weniger Kosten und schnellerer Fertigstellung. Qualitätsarbeit motiviert dagegen die Projektbeteiligten. Sie sind stolz auf ihr Produkt und engagieren sich entsprechend.
- **Babyschritte** (baby steps): Um das Risiko zu verringern, werden Veränderungen in kleinen Schritten durchgeführt. Durch viele kleine Schritte wird dennoch ein hohes Entwicklungstempo erreicht.
- **Akzeptierte Verantwortung** (accepted responsibility): Verantwortung muss akzeptiert werden. Wer sich einer Geschichte (story) annimmt, ist für den Entwurf, die Programmierung und den Test verantwortlich.

XP2 Primärpraktiken

- **Beieinander sitzen** (sit together): Alle Projektbeteiligten sitzen räumlich nahe beieinander - idealerweise in einem Raum.
- **Vollständiges Team** (whole team): Die Teammitglieder verfügen über das vollständige fachliche und technische Know-how, um unabhängig von externen Ressourcen arbeiten zu können.
- **Informative Arbeitsumgebung** (informative workspace): Die

  Arbeitsumgebung des Teams soll über den aktuellen Projektzustand informieren - über offene Aufgaben, über den Testzustand usw. Dies kann durch Karten an Pinwänden, durch Ausdrucke an den Wänden des Teamraums usw. geschehen.
- **Energiegeladene Arbeit** (energized work): Alle Teammitglieder sind engagiert bei der Arbeit.
- **Paarweises Programmieren** (pair programming); Unverändert gegenüber XP.
- **Geschichten** (stories): Die Anforderungen werden als informelle Geschichten aufgeschrieben.
- **Wochenzyklus** (weakly cycle): Jede Iteration dauert eine Woche.

- **Quartalszyklus** (quarterly cycle): Jedes Release dauert drei Monate
- **Freiraum** (slack): Entwickler erhalten während des Projekts Zeit, um sich weiterzubilden.
- **Zehn-Minuten-Build** (ten-minute build): Das Erstellen einer unvollständigen und vorübergehenden, aber ausführbaren Version des in Entwicklung befindlichen Systems darf nicht länger als zehn Minuten dauern. Durch geschickte Komponentenaufteilung ist dies auch bei großen Projekten möglich.

- **Fortlaufende Integration** (continuous integration): Unverändert gegenüber XP.
- **Test vor Programmierung** (Test-first programming): Vor der Programmierung werden die Tests erstellt.
- **Inkrementeller Entwurf** (incremental design): Der Entwurf erfolgt schrittweise, getrieben von den aktuellen Anforderungen. In XP hieß diese Praktik »Einfacher Entwurf«, was zu Missverständnissen führte. Der Entwurf darf komplex sein. Ein großer Gesamtentwurf (big design upfront) ist aber nicht erlaubt. Die Re-Strukturierung (refactoring) ist keine eigene Praktik mehr, sondern gehört zum inkrementellen Entwurf dazu.

XP2 11 Folge-Praktiken

- **Echte Kundenbeteiligung** (real customer involvement): Ein »echter« Kunde sollte zum Team hinzukommen, wenn das Team die Arbeit mit Geschichten (stories) und inkrementellem Entwurf beherrscht. In XP hieß die Praktik »Kunde vor Ort«.
- **Inkrementelle Auslieferung** (incremental deployment): Erstellte Releases sollen beim Kunden auch eingesetzt werden, auch wenn nur Teil der Funktionalität verfügbar sind. Sie sollten bereits parallel mit Altsystemen benutzt werden.
- **Team-Kontinuität** (learn continuity): Projektmitarbeiter sollen nur in genau einem Projekt arbeiten. Personalwechsel sollten vermieden werden, da ein Großteil des Projekts sich in den Köpfen der Mitarbeiter befindet.
- **Schrumpfende Teams** (shrinking teams): Die Produktivität wird erhöht, wenn es gelingt Projektmitarbeiter zu 100 Prozent freizusetzen, damit sie in anderen Teams arbeiten können.
- **Ursprungsursachen-Analyse** (root-cause analysis): Wichtige Probleme werden systematisch untersucht, um die Ursache zu finden.
- **Gemeinsamer Code** (shared code): In XP heißt diese Praktik »Gemeinsame Verantwortung«. In XP2 ist es eine Folgepraktik, da sie Gefahren beinhaltet. Fühlt sich ein Team nicht gemeinsam verantwortlich, dann führt gemeinsamer Code schnell zu chaotischen Systemstrukturen.

- **Code und Tests** (code and tests): Nur der Quellcode und die Tests werden als Dokumentation erstellt und gepflegt.
- **Eine Codebasis** (single code base): Es gibt nur eine Codebasis. Zweige (branches) sind nur kurzzeitig erlaubt.
- **Tägliche Auslieferung** (daily deployment): Täglich soll der aktuelle Systemzustand an die Anwender ausgeliefert werden.

- **Vertrag mit aushandelbarem Umfang** (negotiated scope contract): Anstelle eines Festpreisvertrags wird ein Vertrag mit festem Budget und verhandelbarem Funktionsumfang abgeschlossen.
- **Bezahlung pro Benutzung** (pay per use): Das System wird nicht pro Release bezahlt, sondern es werden Funktionen pro Benutzung abgerechnet. Kunde und Auftragnehmer ziehen so an einem Strang. Eine häufige Benutzung der Funktionen erzeugt für den Kunden einen hohen Geschäftswert und gleichzeitig Umsätze für den Auftragnehmer.

XP2 Bewertung

Vorteile

+ In einem dynamischen Umfeld mit nicht im Voraus bekannten Anforderungen, einem innovativen Produkt und einer kleinen bis mittleren Projektgröße reduzieren die Prinzipien und Praktiken von XP2 die Risiken bei einer Softwareentwicklung.
+ Durch die Konzentration auf die Kundenwünsche, auf den lauffähigen Code, auf die hinein entwickelte Qualität und kurze Auslieferungszyklen wird jeder unnötige Ballast vermieden.

Vorteile

- Viele Annahmen insbesondere bzgl. der Qualität, der Dokumentation und der Arbeitsweise sind empirisch nicht belegt. Es gibt gegenteilige Aussagen und Untersuchungen.
- In XP2 werden für einzelne Entwicklungstätigkeiten absolute Zeitraster eingeführt (time-boxed development): Wochenzyklus für Iterationen, Quartalszyklus für Releases. Nachteilig daran ist, dass **kritische Anforderungen oft nicht in solch festen Rastern umgesetzt werden können**. Auch lässt sich eine zu umfangreiche Funktionalität nicht beliebig auf ein festes Zeitraster skalieren. Siehe auch Bewertung zu XP

XP2 Klassifikation

- Prozess- und Qualitätsmodell.
- Mittelgranulares Modell.
- Nur bedingt anpassbares Modell.
- Agiles Modell.
- Softwarespezifisches Modell mit Konzentration auf das technische Management.
- Modell für die Erstellung in einem Team in einem Raum.
- Modell für die Erstellung von Individualsoftware.

2.10 FDD Feature Driven Development

- `http://www.featuredrivendevelopment.com`, von Jeff De Luca `http://www.nebulon.com/`
- iteratives und inkrementelles Prozessmodeli für die
- modellgetriebene, objektorientierte Softwareentwicklung
- im Rahmen eines Bankprojektes in Singapore 1997 entwickelt
- 5 Prozesse
- Prozess 1: Erstellung Gesamtmodell
- Prozess 2: Erstellung einer Feature-Liste
- Prozess 3: Planen jedes Features
- Prozess 4: Entwurf jedes Features
- Prozess 5: Implementieren jedes Features

FDD Prozess 1: Erstellung Gesamtmodell

- Wer: Modellierungsteam: Anwendungsspezialisten (domain experts) und Chefprogrammierer (chief programmers), geleitet von einem Chefarchitekten (chief architect), neuerdings Chefmodellierer (chief modeler) genannt
- Was: Gesamtmodell:
  - Alle Klassen des Anwendungsbereichs und ihre Verbindungen sind identifiziert, aber nicht alle Attribute und Methoden.
    Ziel dieses Prozesses ist es, einen Konsens über den Umfang, den Inhalt und das Gesamtmodell des zu entwickelnden Systems zu erzielen.
    Der Prozess wird mit einem internen oder externen Assessment abgeschlossen.
- Wie:
  - Eine Person oder zwei Personen erstellen jeweils ein Fachmodell für einzelne Domänen-Bereiche.
  - Der Chefarchitekt kann ein Strohmann-Modell (strawman model) vorschlagen,
    um die Arbeit der Gruppen zu erleichtern.
  - Ein Mitglied jeder Gruppe stellt die erarbeiteten Teilmodelle vor.
  - Der Chefarchitekt kann
    weitere Modellalternativen vorschlagen.

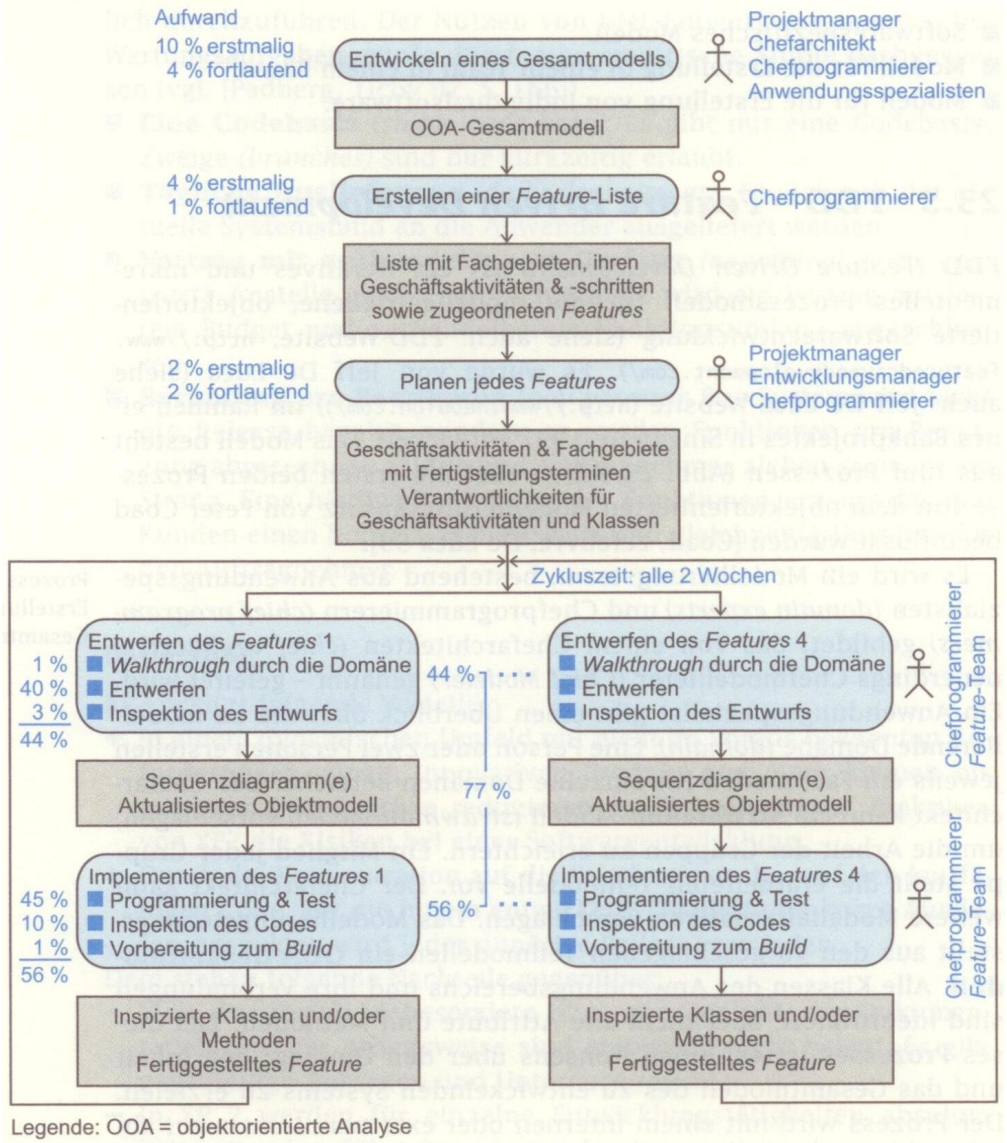


Bild 2.48: Überblick über das FDD-Prozessmodell, aus [Bal08]

* Das Modellierungsteam erstellt aus den vorgeschlagenen Teilmodellen ein Gesamtmodell.

FDD Prozess 2: Erstellung einer Feature-Liste

- Wer: Chefprogrammierer
- Was: Liste mit Fachgebieten (subject areas) ihren Geschäftsaktivitätsschritte (business activity steps) sowie zugeordnete Features (Merkmale) Geschäftsaktivitätsschritte werden durch Features realisiert.
- **Features** sind feingranulare Funktionen, die für den Auftraggeber einen Wert darstellen.

- beschrieben nach Schema:

  <Aktion> <Ergebnis> <Objekt>

- Jedes Feature soll in maximal zwei Wochen realisiert werden können, andernfalls wird der Geschäftsaktivitätsschritt weiter unterteilt.

- Abschluss: internes oder externes Assessment

FDD Prozess 3: Planen jedes Features

- Wer: Planungsteam

- Was: Geschäftsaktivitäten und Fachgebiete mit Fertigstellungsterminen Verantwortlichkeiten für Geschäftsaktivitäten und Klassen

- Wie:

  Ein Planungsteam legt fest, in welcher Reihenfolge die Features

  realisiert werden sollen. Für die Geschäftsaktivitäten. werden Fertigstellungstermine (Monat und Jahr) bestimmt. Bei der Planung werden Faktoren wie Abhängigkeiten zwischen den Features, Risiken, Arbeitsauslastung der Entwicklungsteams sowie Meilensteine

  des Auftraggebers berücksichtigt. Jeder Geschäftsaktivität wird ein Chefprogrammierer zugeordnet, der dafür die Verantwortung übernimmt. Jedem Entwickler wird eine Klasse (in Ausnahmefällen auch mehrere Klassen) zugeordnet (class owner). Ein Klassen-Verantwortlicher ist für den Entwurf und die Implementierung seiner Klasse zuständig. Nur er darf die Klasse ändern.

- Abschluss: Self-Assessment

FDD Prozess 4: Entwurf jedes Features

- Wer: jeweils Chefprogrammierer und temporäres Feature-Team

- Was: objektorientierte Sequenzdiagramme und Aktualisiertes Objektmodell

- Wie: Die Chefprogrammierer stellen temporäre Feature-Teams auf der

  Basis der zugeordneten Klassenverantwortlichkeiten zusammen, die

  nicht mehr als zwei Wochen zusammenarbeiten.

  Ein Domänenexperte gibt einen Überblick über die Domäne, zu dem das Feature gehört.

  Jedes Feature-Team erstellt ein oder mehrere objektorientierte Sequenzdiagramme für das jeweilige Feature. Die Chefprogrammierer

  verfeinern die Klassenmodelle anhand der Sequenzdiagramme.

Die Entwickler erstellen erste Klassen- und Methodenrümpfe.

Die Klassen-Verantwortlichen arbeiten parallel in mehreren Feature-Teams mit.

Die Zugehörigkeit zu einem Feature-Team kann mit jeder Iteration wechseln.

Die Interaktionen innerhalb eines Teams

spielen sich im Wesentlichen zwischen dem jeweiligen Teammitglied

und dem Chefprogrammierer ab - weniger untereinander.

- Abschluss: Entwurfsinspektion

FDD Prozess 5: Implementieren jedes Features

- Wer: jeweils Chefprogrammierer und temporäres Feature-Team
- Was: Programmierung und Test des Features, Inspektion des Codes Vorbereitung zum Build
- Wie: Die Entwickler implementieren die im Prozess 4 vorbereiteten Features. Zur Qualitätssicherung werden Komponententests und Code-Inspektionen eingesetzt.

Vergleich FDD - XP

- Im Gegensatz zu XP gibt es **für jede Klasse einen verantwortlichen Entwickler** *(Class-Ownership)*, während in XP jeder im Team jede Klasse ändern kann *(Collective-Ownership)*.
- FDD verwendet **formale Inspektionen** (die als wirksamer angesehen werden), statt paarweises Programmieren wie in XP
- In FDD gibt es einen **Chefarchitekten und Chefprogrammierer**. Sie verfügen über mehr Erfahrung und sind gleichzeitig Mentoren für die Entwickler. In XP gibt es keine Rollen.
- In FDD legt der Auftraggeber die Prioritäten bei den Geschäftsaktivitäten fest,

  während die Feature-Prioritäten vom Chefprogrammierer nach technischen Gesichtspunkten bestimmt werden. In XP legt der Auftraggeber die Prioritäten fest.
- In FDD zunächst ein **grobes Gesamtmodell** erstellt, um Überarbeitungsarbeiten (rework) durch spätere Erkenntnisse zu vermeiden (»doing as much right the first time«). In XP nicht.
- In FDD nur begrenztes Re-Strukturieren. In XP viel Re-Strukturieren.

FDD Bewertung

- inkrementellem Basismodell
- \+ Gut geeignet für die objektorientierte Neuentwicklung von Individual- und Standardsoftware
- \+ Allg. Parallelarbeit mehrerer Teams in den Prozessen 4 und 5.
- \- Sequentielle Abfolge der Prozesse. Ein freier Wechsel zwischen den Prozessen ist nicht vorgesehen

---

FDD Klassifikation

- Prozess- und Qualitätsmodell.
- Kein reines Prozess- oder Qualitätsmodell, sondern integriert eine Methode für die objektorientierte Entwicklung.
- Spezialisierung des inkrementellen Basismodells für eine objektorientierte Softwareentwicklung.
- Agiles Modell.
- Mittel- bis feingranulares Modell.
- Modell speziell für objektorientierte Softwareentwicklungen.
- Modell für die Erstellung von Individualsoftware oder Standardsoftware.

2.11 Scrum

`http://www.scrumalliance.org/`
Einführung von Mike Cohn, `www.mountaingoatssoftware.com`
Lizenz creative commons

Scrum in 100 Worten

- Scrum ist ein agiler Prozess, der es erlaubt auf die Auslieferung der wichtigsten Geschäfts-Anforderungen innerhalb kürzester Zeit zu fokussieren.
- Scrum gestattet es, schnell und in regelmäßigen Abschnitten (von zwei Wochen bis zu einem Monat) tatsächlich lauffähige Software zu inspizieren.
- Das Business setzt die Prioritäten. Selbst-organisierende Entwicklungsteams legen das beste Vorgehen zur Auslieferung der höchstprioren Features fest.
- Alle zwei Wochen bis zu einem Monat kann jeder lauffähige Software sehen und entscheiden, diese so auszuliefern oder in einem weiteren Abschnitt zu ergänzen.

Bild 2.49: Scum beim Rugby, Originalfoto von `http://zu.wikipedia.org/wiki/Umbhoxo`

Scrum, Bewertung nach Balzert

- Scrum legt einen Projektrahmen für kleine Teams fest.
- Von der Zielsetzung her vergleichbar mit dem Spiralmodell

+ Gut geeignet, wenn die Anforderungen nicht im Voraus festgelegt werden können und »chaotische« Rahmenbedingungen antizipiert werden müssen.

+ Leicht erlernbar und schnell einsetzbar; auf ein paar Seiten beschreibbar.

- Erfolg hängt stark von den Teammitgliedern und ihrer Persönlichkeit ab.

- Keine Hilfestellungen für die Entwicklung von Software.

Scrum Klassifikation

- Reines Prozessmodell Art und Weise der Qualitätskontrolle legt das Team fest oder auch nicht.
- Grobgranularer Rahmen mit feingranularen Festlegungen in einigen Details (30 Tage Sprint, 15 Minuten tägliche Besprechung).
- Allgemeines Modell.
- Modell für die Erstellung in einem Team.

Scrum-Charakteristika

Scrum ist einer von mehreren existierenden sogenannten agilen Prozessen für Softwareentwicklung und Projektmanagement - andere sind z.B. Crystal, Extreme Programming (XP), Feature Driven Development (FDD).

Scrum bietet eine schlanke Projektmanagement-Methode mit folgenden Charakteristika:

- einfache Regeln
- wenige Rollen
- mehrere Arten von Meetings mit bestimmten Zwecken
- einige Schlüssel-Artefakte, deren Pflege Overhead vermeidet und die maximale Transparenz auf einfache Weise bieten
- Pragmatismus statt Dogmatik
- iteratives Vorgehen
- Selbstorganisation und Eigenverantwortung in interdisziplinären Teams
- Konzentration auf hochqualitative Arbeit anstatt auf eine Papierflut bei der Spezifikation
- Änderungen der Kundenanforderungen während des Projekts gelten als normal, nicht als Störfaktor (es gibt keine "fertige" Spezifikation)
- speziell geeignet für hochkomplexe Projekte mit unklaren Anforderungen

---

Historisches zu Scrum

- Bereits Mitte der 1980er Jahre gab es erste Tendenzen, die bekannten Projektmanagement-Methodiken in Frage zu stellen und nach agileren Ansätzen zu suchen.
- Seit den 1990er Jahren werden Scrum-Projekte durchgeführt.
- Etwa seit der Jahrtausendwende war der Erfolg von Scrum mit der explosionsartig wachsenden Zahl erfolgreicher Projekte nicht mehr aufzuhalten.

Harvard BusinessReview schrieb 1986 in "The New New Product Development Game":

- Die (...) "Staffellauf-Vorgehensweise" in der Produktentwicklung (...) gerät unter Umständen in Konflikt mit dem Ziel maximaler Geschwindigkeit und Flexibilität. Stattdessen könnte ein ganzheitlicher "Rugby-Ansatz" - in welchem ein Team versucht, Wege als Einheit zurückzulegen und dabei den Ball hin- und herspielt - den Anforderungen des heutigen Wettbewerbs dienlicher sein.

Weitere Meilensteine:

- "Wicked Problems, Righteous Solutions" von DeGrace und Stahl erwähnt 1990 erstmals Scrum im Zusammenhang mit Software.
- Jeff Sutherland führt 1993 erste Scrums bei Easel Corp. durch.
- Ken Schwaber liefert bei der OOPSLA 96 gemeinsam mit Sutherland eine erste Definition von Scrum
- Scrum etabliert sich innerhalb weniger Jahre in tausenden Projekten mit bis zu mehreren hundert Beteiligten und wird als konform zu FDA-, ISO-9000- und anderen Standards anerkannt.

Scrum - auf einer Seite erklärt

- Scrum kennt drei Rollen für direkt am Prozess Beteiligte:
  - **Product Owner** (stellt fachliche Anforderungen und priorisiert sie),
  - **ScrumMaster** (managt den Prozess und beseitigt Hindernisse) und
  - **Team** (entwickelt das Produkt).
  - Daneben gibt es als Beobachter und Ratgeber noch die **Stakeholders**.

- Die Anforderungen (**Requirements**) werden in einer Liste (**Product Backlog**) gepflegt, erweitert und priorisiert.
  - Das Product Backlog ist ständig im Fluss.
  - Um ein sinnvolles Arbeiten zu ermöglichen, wird monatlich vom Team in Kooperation mit dem Product Owner ein definiertes Arbeitspaket dem oberen, höher priorisierten Ende des Product Backlogs entnommen und komplett in Funktionalität umgesetzt (inkl. Test und notwendiger Dokumentation).
  - Dieses Arbeitspaket, das **Increment**, wird während der laufenden Iteration, des sog. **Sprints**, nicht durch Zusatzanforderungen modifiziert, um seine Fertigstellung nicht zu gefährden.
  - Alle anderen Teile des Product Backlogs können vom Product Owner in Vorbereitung für den nachfolgenden Sprint verändert bzw. neu priorisiert werden.

- Das Arbeitspaket wird in kleinere Arbeitspakete (**Tasks**) heruntergebrochen und mit jeweils zuständigem Bearbeiter und äglich aktualisiertem Restaufwand in einer weiteren Liste, dem **Sprint Backlog**, festgehalten.

  - Während des Sprints arbeitet das Team konzentriert und ohne Störungen von außen daran, die Tasks aus dem Sprint Backlog in ein **Increment of Potentially Shippable Functionality**, also einen vollständig fertigen und potentiell produktiv einsetzbaren Anwendungsteil, umzusetzen.

  - Das Team gleicht sich in einem täglichen, streng auf 15 Minuten begrenzten Informations-Meeting, dem **Daily Scrum Meeting**, ab, damit jeder weiß, woran der andere zuletzt gearbeitet hat, was er als nächstes vor hat und welche Probleme es evtl. gibt.

- Am Ende des Sprints präsentiert das Team dem Product Owner, den Stakeholders u.a. interessierten Teilnehmern in einem sog. **Sprint Review Meeting** live am System die implementierte Funktionalität.

  - Halbfertiges oder gar Powerpoint-Folien sind während des Reviews verboten.

  - Das Feedback der Zuseher und die neuen Anforderungen des Product Owners für den kommenden Sprint fließen dann wieder in das nächste Sprint Planning Meeting ein, und der Prozess beginnt von neuem.

Scrum-Prozess

- Der ScrumMaster sorgt während des gesamten Prozesses dafür, dass Regeln eingehalten werden und der Status aller Tasks im Sprint Backlog von den jeweils zuständigen Team-Mitgliedern täglich aktualisiert wird.

  - Er macht den Projektfortschritt transparent durch einen geeigneten Reporting-Mechanismus: die Veröffentlichung sog. **Burndown Charts**, welche den Fortschritt für den aktuellen Sprint bzw. für das gesamte Projekt jeweils in Form einer Kurve visualisieren. Eingezeichnete Trendlinien erlauben es, mögliche Probleme und Verzögerungen einfach (und rechtzeitig!) zu erkennen.

- Im Kern basiert Scrum also auf einer inkrementellen Vorgehensweise, der Organisation von Entwicklungsabschnitten und Meetings in vordefinierten Zeitabschnitten (**Time-Boxes**) und der Erkenntnis, dass ein funktionierendes Produkt wichtiger ist als eine dreihundertseitige Spezifikation.

2.12 Monumental - Agil

Quervergleich wesentlicher Charakteristika monumentaler und agiler Prozessmodelle

Monumentales Modell	**Agiles Modell**
Prozess vorhersagbar	Prozess adaptiv
prozessorientiert	menschen- und teamorientiert
formale Kommunikation	informelle Kommunikation
umfangreiche, formale Dokumentation	minimale, informale Dokumentation
keine festen Zeitraster	oft feste Zeiraster (*time box*)
mittelmäßig qualifizierte, z.T. unmotivierte EntwicklerInnen	verantwortungsvolle und motivierte EntwicklerInnen
KundInnen, die wenig mit der Entwicklung zu tun haben wollen	KundInnen, die in die Entwicklung einbezogen werden wollen
Teamgröße über 50	Teamgröße kleiner 50
Festpreisauftrag	Auftrag nach Aufwand