

Hinweis:

Diese Druckversion der Lerneinheit stellt aufgrund der Beschaffenheit des Mediums eine im Funktionsumfang stark eingeschränkte Variante des Lernmaterials dar. Um alle Funktionen, insbesondere Animationen und Interaktionen, nutzen zu können, benötigen Sie die On- oder Offlineversion. Die Inhalte sind urheberrechtlich geschützt.
©2018 Beuth Hochschule für Technik Berlin

VOR - Vorgehensmodelle / Agile Modelle



Lernziele und Überblick

Vorgehensmodelle sind mit der Vogelperspektive vergleichbar. Sie geben einen hilfreichen Überblick über das gesamte Projekt. Nachdem das „**was**“ geklärt worden ist, muss sich mit dem „**wie**“ auseinander gesetzt werden, genauer gesagt, wie soll das Projekt organisiert und wie soll generell vorgegangen werden. Es soll auch die Frage beantwortet werden: „Was machen wir wann?“ Dabei wird in der Regel festgelegt, in welcher Zeit, welche Ressourcen welche Arbeit tun und welche Artefakte (z. B. Dokumente oder Programme) zu erstellen sind.

Vorgehensmodelle können unterteilt werden in die Kategorien Alt, Schwergewichtig und Agil.

1. **Alte** Vorgehensmodelle haben fast nur noch historischen Wert.
2. Eher „**schwergewichtige**“ Vorgehensmodelle; haben ebenfalls eine längere Historie und sind - mehr oder weniger - hinsichtlich Agilität aktualisiert worden.
3. Agile Modelle - agil steht in diesem Zusammenhang für etwas Bewegliches, Lebhaftes, oder Wendiges.

Diese Lerneinheit dient als Einführung, es werden relevante Begrifflichkeiten geklärt und Modelle zueinander in Beziehung gesetzt. Der Umgang und Gebrauch von Vorgehensmodellen in aktuellen Projekten ist Bestandteil dieser Lerneinheit.



Hinweis

Praktische Erfahrung mit Vorgehens- oder Agilen Modellen sind vorteilhaft und bilden eine gute Wissensgrundlage für eine Vertiefung. Es ist daher anzuraten, sich bei eigenen Software- oder Übungsprojekten Vorgehensweisen oder gute Ideen zu dokumentieren und diese anzuwenden. Auf diese Weise gewinnt man die Erfahrung, wie Vorgehensmodelle am geschicktesten und am passendsten genutzt werden können!



Gliederung

Zu Beginn werden Begriffe zu Vorgehensmodellen und zu wesentlichen Elementen vorgestellt. Diese werden durch einen anschließenden Blick in die Historie abgerundet.

Danach folgen Wasserfall- und Spiralmodelle sowie die Merkmale agiler Modelle.

In weiteren Kapiteln werden industrierelevante und umfangreichere Modelle wie der Rational Unified Process (RUP) und das V-Modell vorgestellt.

Abschließend werden die leichtgewichtigeren und agilen Modelle betrachtet. XP wird dabei etwas ausführlicher dargestellt, Crystal Clear sowie SCRUM können nur kurz angerissen werden.



Lernziele

Die vorgestellten Vorgehensmodelle sollen in ihren Grundzügen und Prinzipien bekannt sein und eingeordnet werden können. Der Student soll in der Lage sein, entscheiden und abwägen zu können, wann welches (bestimmtes) Vorgehensmodell besser geeignet ist als ein anderes. Dabei sind zu unterscheiden:

- Grundbegriffe: Vorgehensmodell, Prozessmodell
- Historie, Wasserfallmodell
- Rational Unified Process (RUP)
- Object Engineering Process (OEP)
- Das V-Modell und das **V-Modell XT**

und agile Modelle wie

- Extreme Programming (XP)
- **SCRUM**
- **Crystal Clear**



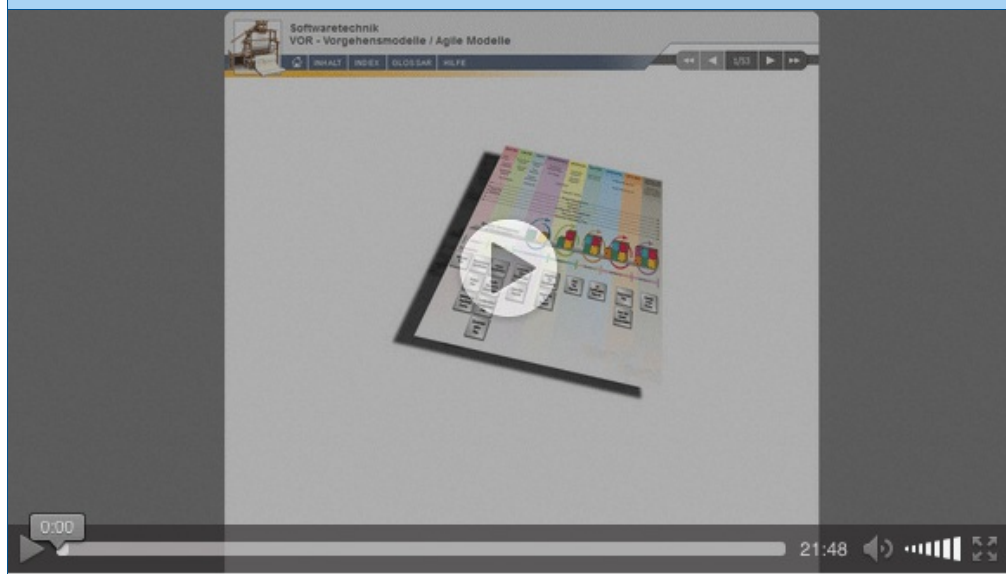
Zeitbedarf und Umfang

Für die Bearbeitung dieses Modules benötigen Sie 180 Minuten. Für die Übung benötigen Sie nochmals 130 Minuten.



Film

Webkonferenz zur Lerneinheit VOR



© Beuth Hochschule Berlin - Dauer: 21:47 Min. - Streaming Media 40.2 MB

Die Hinweise auf klausurrelevante Teile beziehen sich möglicherweise nicht auf Ihren Kurs. Stimmen Sie sich darüber bitte mit ihrer Kursbetreuung ab.

1 Einführung

Worum geht es?

Mit Hilfe von Vorgehensmodellen sollen in der Softwaretechnik die folgenden Fragen beantwortet werden:

- In welche **Teile** kann der Prozess gegliedert werden?
- In welcher **Reihenfolge** können die Teilschritte ausgeführt werden?
- Welche Ressourcen (Menschen, Informationen) sind erforderlich?
- Welche Informationen müssen als Eingangsgrößen vorliegen?
- Welche **Ergebnisse** werden von welchen **Teilschritten** erzeugt?
- Welche **Modellierungstechnik** ist für welche Prozessleitschritte sinnvoll einsetzbar?

Letzteres ist besonders im OO-Kontext wichtig, weil bei einigen Projekten gezielt darauf geachtet werden muss wiederverwendbaren Code und Komponenten zu erstellen. Bei anderen Projekten hingegen - z. B. bei Projekten mit funktionalen Programmiersprachen - ist dies weniger von Bedeutung.

Also was , wie und womit !

1.1 Begriffsdefinitionen

Bei der Betrachtung müssen die Begriffe Vorgehensmodell, Prozessmodell und Prozessleitfaden unterschieden werden.

Das Vorgehensmodell ist die abstrakte und übergeordnete Antwort auf die Fragen, was, wie und womit durchzuführen ist. Schauen wir uns eine Definition an:



Definition

Vorgehensmodell

„Ein **Vorgehensmodell** gliedert den Prozess des Organisierens in verschiedene, strukturierte Phasen, denen wiederum entsprechende Methoden und Techniken der Organisation zugeordnet sind. Aufgabe eines Vorgehensmodells ist, die allgemein in einem Gestaltungsprozess auftretenden Aufgabenstellungen und Aktivitäten in ihrer logischen Ordnung darzustellen.

[...]

Gemeinsam ist allen Vorgehensmodellen der **schrittweise Weg** vom Problem zur Lösung und ihr systematisch **rationales Vorgehen** (im Gegensatz etwa zu **Versuch und Irrtum**, siehe Entscheidungsstil). Die einzelnen Phasen sind Idealtypen. In der Praxis ist es oft notwendig, iterativ vorzugehen und ‚zurückzuspringen‘. Phasenorientierte Meilensteine sollen das Risiko und Kosten eines Scheiterns minimieren. “

(Quelle: www.wikipedia.de)

Prozessmodell

Häufig wird in einem leicht abweichenden Zusammenhang von einem Prozessmodell gesprochen, welches sich eher auf eine fachliche Beschreibung von Geschäftsprozessen bezieht. Angewendet auf das Vorgehen, bei der Erstellung von Software, innerhalb eines Projekts beschreibt ein Prozessmodell die Schritte und Tätigkeiten die im Projekt anfallen. Es beschreibt nicht die Geschäftsprozesse im klassischen Sinne eines „Firmengeschäftes“.

Prozessleitfaden

Betrachtet man weiterhin die Vorgehensweise im Softwareprojekt so ist der Prozessleitfaden ein Dokument, welches aus dem Vorgehensmodell abgeleitet und angepasst wird. Es liefert konkrete Schritte und Anleitungen für das Projekt, wohingegen das Vorgehensmodell eher abstrakt bleibt.

Ein typischer Vorgang zum Projektstart ist das Vorgehensmodell so anzusetzen und zu konkretisieren, dass es passt. Natürlich in Bezug auf die Firma und das zu lösende Problem. Diesen Vorgang nennt man „**Tailoring**“. Das Vorgehensmodell oder die Vorgehensmodelle werden maßgeschneidert - es werden die passenden Dinge herausgenommen und an das Projekt angepasst.

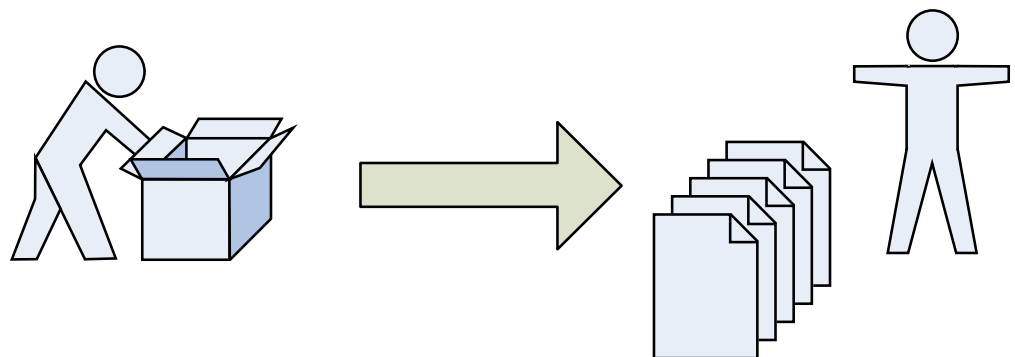



Abb.: Tailoring

Vorgehensmodell

passendes Prozessmodell

1.2 Vorgehensmodelle allgemein

Die aus den Vorgehensmodellen abgeleiteten Prozessleitfäden liefern konkrete Schritte und Anleitungen. Typischerweise enthalten sie die Beschreibung der folgenden Punkte:

- **Phasen:** Zeitabschnitte werden festgelegt oder umrissen, in denen bestimmte gleiche Aktivitäten angesiedelt sind. So könnte es beispielsweise explizit eine Phase für die Qualitätssicherung geben.
- **Iterationen:** Hier steht die Frage, wann und wie ein Rücksprung stattfinden darf bzw. wie sich Schritte wiederholen dürfen.
- **Meilensteine:** Welche wichtigen Projekt- oder Softwarezustände sollen wann erreicht werden? Das Formulieren von Meilensteinen ist aus dem Projektmanagement hinreichend bekannt und wird durch viele Tools unterstützt. Wichtig ist die verbindliche Festschreibung. 
- **Aktivitäten:** Beschreibung der Tätigkeiten, die im Rahmen des Projektes durchgeführt werden müssen. z. B. ein Interview einer Person, Schreiben eines Reportes, Entwicklung einer Komponente oder dem Test eines Moduls.
- **Prozesse:** Im Vordergrund steht hier die zeitliche und logische Abfolge von Aktivitäten. Dies stellt eine Art Workflow dar.
- **Akteure:** Welchen Fähigkeiten und Aufgabenbereichen sind notwendig und welchen Personen können diese zugeordnet werden.
- **Ergebnisse:** Welche Ergebnisse (z. B. Dokumente) müssen nach welchen Phasen vorhanden sein?
- **Dokumente:** Welche Dokumente müssen als Ergebnis nach einer Phase vorliegen? Für Dokumente oder sonstige Ergebnisse wird auch der Begriff Artefakte verwendet - aus dem lateinischen „factum“ = das Gemachte.

Praxiserfahrungen bei Vorgehensmodellen:

Theorie machbar

➕ Das heutige **Wissens** zu Vorgehensmodellen ist in großen Teilen bereits seit den 90er Jahren präsent und kommt dem Betrachter sicherlich **bekannt** vor. Insbesondere in Bezug auf die Themen Objektorientierung und Agilität gibt es jedoch einige neue Ansätze, die sich auch in der Technikunterstützung widerspiegeln.

➕ Letztes bedeutet eigentlich auch, dass die ingenieurmäßige **Anwendung** der Vorgehensweisen aus Vorgehensmodellen **theoretisch kein Problem** darstellt.

Praxis immer problematisch

➖ In der Praxis sieht leider vieles anders aus. Insbesondere weil die Ressourcen, wie Geld und Zeit immer beschränkt sind. Allzu oft heißt es daher: „Ja, wir wenden ‚im Prinzip‘ ein Vorgehensmodell wie den RUP an“ oder aber „wir haben für bestimmte Vorgaben wie Qualitätssicherung, Dokumentation etc. keine Zeit“. Auch dieser Aspekt soll hier betrachtet werden.

➖ Das „**Tailoring**“ von Vorgehensmodellen kostet ebenfalls viel Zeit und **Geld**, wovon leider nie ausreichend zur Verfügung steht. Jede Firma sollte zwar ein angepasstes Modell aus der Tasche zaubern können (Marke: wir können und würden RUP-LIGHT-COMPANY-X), ohne dabei jedoch auf eine individuelle Anpassung zu achten und ohne die nötige Ruhe und Reife in den Tailoring-Prozess einfließen zu lassen.

➖ Tatsächlich bleibt in den meisten Fällen ein Prozessleitfaden oder Praxisleitfaden auf der Strecke. Das Modell des angepassten Prozesses ist dann meist einzig und allein im Kopf des Projektleiters verankert und somit kann keine außenstehende Kontrolle oder Anpassung vorgenommen werden. Was den Mitarbeitern und Entwicklern bleibt, sind Leitfäden für Kodierungsstandards und selten mehr.

1.3 Historie

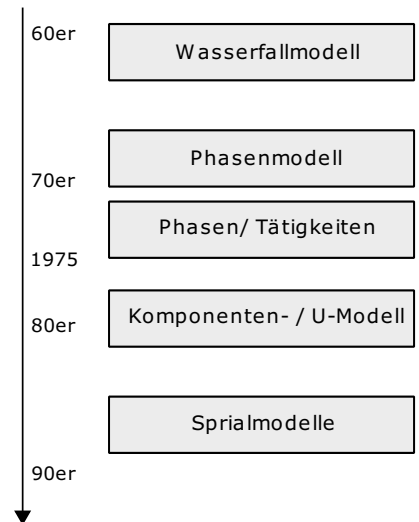
Ein Blick auf die Historie zeigt, dass sich in den ersten Jahren bei größeren Softwareprojekten eine **Planungseuphorie** durchsetzte.

Anfangs war man der Meinung, alles genau zu planen und vorher als auch nachher alles genau dokumentieren zu können. Man musste also einfach nur alles vorher lange genug und extrem ausführlich **spezifizieren** und dann klappt das schon...

Dieser Ansatz führte zu den ersten Modellen, die heutzutage - etwas verächtlich - **Wasserfallmodelle** genannt werden.

Das Department of Defense (DOD) hat noch bis in die späten 80er Jahre an Modellen festgehalten, die den Wasserfallmodellen entsprachen.

Später kamen dann doch erste Modelle auf, in denen iterativ oder in Zyklen gearbeitet wurde.



1.4 Wasserfallmodell

Das Wasserfallmodell ist der Klassiker unter den Modellen. Es beschreibt den Ablauf des Entwicklungsprozesses in der Weise, dass die Ergebnisse einer Phase in die nächste Phase fallen - wie bei einem Wasserfall. Die Abbildung zeigt das Wasserfall-Modell mit den klassischen Phasen.

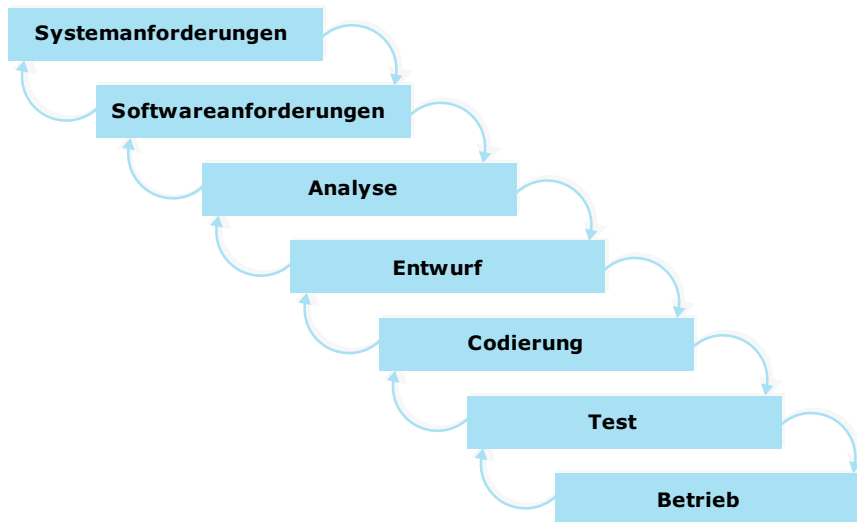


Abb.: Starrer Ablauf im Wasserfallmodell

Das Wasserfallmodell wurde nach und nach an die praktischen Erfordernisse angepasst und erweitert, so konnte es sich bis heute als klassisches Vorgehensmodell in der Software-Entwicklung für die Industrie und Behörden (stark) verbreiten. Eine Ursache dafür liegt in der Einfachheit und Verständlichkeit des Modells, das mit nur wenig Managementaufwand zu handhaben ist.

Kennzeichen des Modells

Die wesentlichen Merkmale des Wasserfallmodells sind, dass

- die Aktivitäten in der **vorgegebenen Reihenfolge** auszuführen sind.
- am Ende jeder Aktivität ein fertiggestelltes **Dokument** vorliegt.
- das Modell auf einem Top-Down Vorgehen basiert.

Prima planbar

Positive Merkmale des Wasserfallmodells

- ➕ **Aufgaben** und **Phasen** sind **klar** umrissen
- ➕ Das Projekt ist verständlich und leicht zu **managen**
- ➕ **Planung** und Überwachung fallen (scheinbar) leichter

Projekt passen oft nicht in ein Schema

Offensichtliche Nachteile

- ➖ **Projektrisiken** werden spät erkannt!
- ➖ Eine **Rückkehr** in vorige Phasen ist **nicht** so einfach möglich. Die Folge ist in der Regel ein größerer Zeitverzug!
- ➖ Das Projekt ist stark **dokumentationsgetrieben**. Es besteht die Hoffnung, dass eine genügend umfangreiche Spezifikation und Dokumentation das Projekt in die richtigen Bahnen lenkt.

Das Wasserfallmodell ist also sowohl mit Bewunderung als auch mit Skepsis zu betrachten. In vielen Bereichen kann es einfach das passende Modell sein, wenn z. B. die Anforderungen klar sind. In der Regel sind Anforderungen nicht eindeutig und verändern sich außerdem während der Entwicklung. In diesen Fällen sind iterative oder beweglichere Modelle besser geeignet.

1.5 Spiralmodelle

Spiralmodelle wurde gegen Ende der 80er und Anfang der 90er Jahre populär. Bei einem Spiralmodell ist erstmalig ein Durchlaufen der vier Phasen erlaubt, nachdem man einmal alle vier Phasen an einem kleinen Teilbereich vervollständigt hat. Hier kann also die Planung im zweiten und n-ten Durchlauf verbessert und auf Änderungen - wenn auch weiterhin recht spät - reagiert werden.

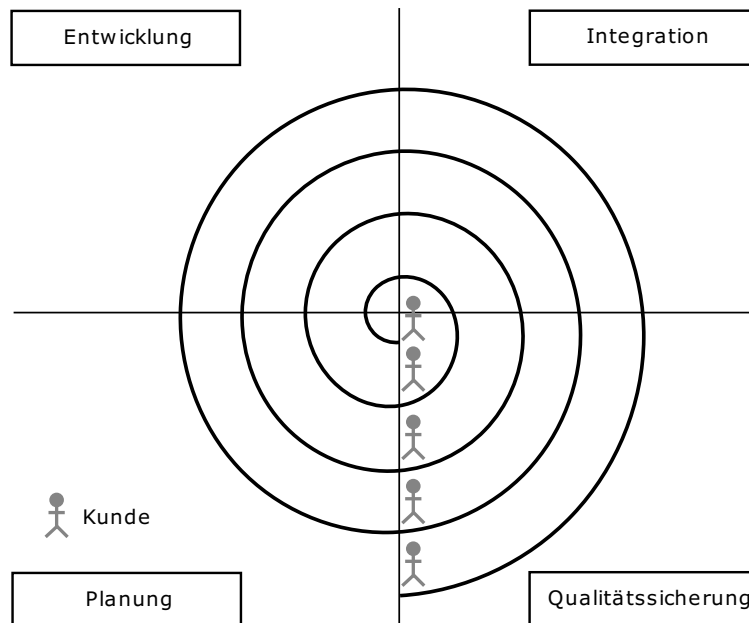


Abb.: Abläufe im Spiralmodell

Inkrementelle Softwareentwicklung

Spiralmodelle bildeten schnell die Basis der **inkrementellen Softwareentwicklung**. Viele Firmen haben auf dieser Basis eigene Modelle entwickelt und diese dann auf bestimmte Branchen (z. B. Bankwesen) zugeschnitten.

Merkmale von Spiralmodellen

- Zeitdruck geht zu Lasten des **Funktionsumfangs**.
- Die Vorgabe fester Zeitrahmen wird als **Time-Boxing** betrachtet.
- Weisen in die Richtung einer evolutionären Softwareentwicklung



1.6 Moderne Methoden / Modelle

Moderne Methoden zeichnen sich zunächst einmal durch eine „**inkrementell-iterative Vorgehensweise**“ aus. Die beim Wasserfall genannten Nachteile wie **Projektrisiken** sollen beseitigt werden. Probleme sollen früher erkannt und schneller behoben werden.

Es wird gerne von einer **evolutionären Vorgehensweise** gesprochen. So wie Lebewesen Generationszyklen durchlaufen und sich anpassen und möglicherweise verbessern, passt sich das Projekt mit seinen Iterationen an die Gegebenheiten an.

Erzeugt Iteration Unsicherheit?

Die Problematik bei iterativen Modellen ist interessanterweise, dass sie zu Beginn Unsicherheit erzeugen können. Beim Wasserfallmodell war alles so schön planbar. Hier aber tauchen immer wieder Fragen auf.

- Wenn alles immer wieder von vorne beginnt, wann sind wir fertig?
- Wie viele Iterationen benötigen wir eigentlich?
- Wie können wir das Planen?
- Wie können wir dies in den Verträgen, den Milestones und den Zeitvorgaben berücksichtigen?



1.7 Vorteile agiler Modelle

Agiler Hype

Die Informatikwelt wurde ab dem Jahr 2000 von einer **Agilen-Welle** überrollt. Das Thema wurde auch in unzähligen Publikationen und auf Konferenzen behandelt. Manifestiert hat sich die Denkweise im bekannten „*Manifesto for Agile Software Development*“, dessen Grundsätze man sich ruhig einmal durchlesen kann.

www.agilemanifesto.org

Mittlerweile hat sich bei einem breiten Kreis von „Vorgehensmodell-Profis“ die Auffassung durchgesetzt, dass beweglichere, also **agilere Modelle** den „alten“ Modellen in der Regel überlegen sind.

Dennoch machen agile Modelle Sinn!

Hinter agilen Modellen stecken handfeste Fakten und Vorteile.

- **Besseres Risikomanagement:** Tritt ein Problem auf, so wird dies durch einen einfacheren Rücksprung in frühere Phasen beseitigt.
- **Fortlaufende Planung:** Die Planung passt sich mehr dem Projekt an und nicht das Projekt der Planung.
- **Kontinuierliche Integration:** Integrationsrisiken können früh erkannt werden.
- **Sichtbarer Projektfortschritt:** Es kann bereits früh ein Prototyp erstellt werden und es gibt somit die Möglichkeit früher ein Feedback zu erhalten.
- **Teststrategien:** Diese sind ein Teil des Projektes und sorgen bereits frühzeitig für mehr Qualität und helfen dabei teure Fehler eher zu erkennen und zu vermeiden.
- **Optimierung des Prozesses:** Falls im Vorgehensmodell oder im Projekt etwas insgesamt nicht passt, so kann es leichter angepasst werden.
- **Flexible Auslieferung:** Teile des Projektes können früher ausgeliefert und somit eher vom Endnutzer getestet werden.

Die Abbildung veranschaulicht den Risikoaspekt und zeigt, dass bei agilen Modellen früher integriert und damit Anpassungen schneller vorgenommen werden können.

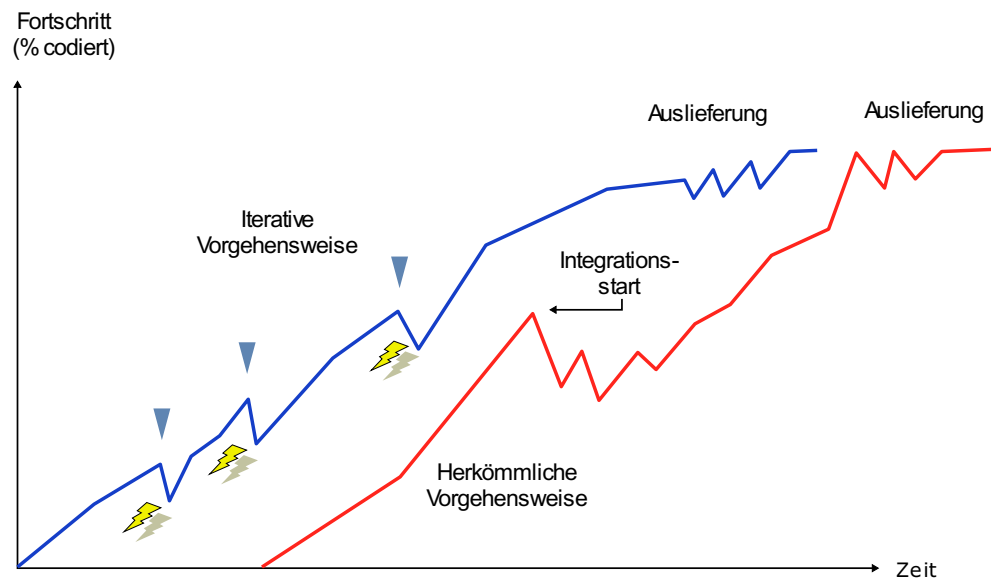


Abb.: Frühe Tests, Integrationen und Prototypen tun zwar weh, aber lenken in die richtige Richtung

2 Der Rational Unified Process (RUP)

RUP wird in dieser Lerneinheit nur kurz betrachtet. Er ist Bestandteil des Studienmoduls „Modellbasierte Softwareentwicklung“. Unterschieden wird zwischen dem Vorgehensmodell des **RUP** und dem des **RUPP**. Letzteres ist das „Rational Unified Process Product“ und wird von der Firma Rational / IBM vermarktet.



IVAR JACOBSON



GRADY BOOCH



JAMES RUMBAUGH

Die Geschichte des Rational Unified Process ist eng mit der Entwicklung von UML (Unified Modeling Language <http://www.uml.org>) verknüpft.

Nach dem Erfolg der ersten iterativen Modelle, war in den 90er Jahren eine stark fragmentierte Welt der Vorgehensmodelle zu erkennen. Es wurden alle möglichen Analysemodelle integriert und „jede“ Firma, sowie „jeder“ Wissenschaftler behauptete von sich, das beste Modell zu besitzen. Problematisch war das Fehlen von **Standards** und somit gab es auch keine Tools, auf denen Hersteller aufsetzen könnten.

IVAR JACOBSON war einer der maßgebenden Kräfte der „großen Vereinheitlichung“. Bis 1987 arbeitete er bei Ericsson und machte sich bereits einen Namen in der Modellierung von Komponenten und „erfand“ die ersten Sequenzdiagramme. 1987 verließ er Ericsson und gründete die Objectory AB. Dabei entstand ca. 1992 der erste „Softwareprozess“ OOSE. 1995 verschmolz die Firma mit Rational Software, wo er mit GRADY BOOCH und JAMES RUMBAUGH zusammenarbeitete, um z. B. UML zu entwickeln.

- 1996 entstand aus der Zusammenarbeit der Rational Objectory Process.
- 1998 entstand der Rational Unified Process 1998 (V 5.0 / 5.5)
- Später dann der Rational Unified Process 2000.
- Im Jahre 2003 kaufte IBM die Firma Rational und integrierte die Produkte teilweise in die IBM Produktfamilie (z. B. IBM Websphere).

So wie das V-Modell von sich behauptet, Standard in Deutschland zu sein (was zu bezweifeln ist), ist der RUP tatsächlich der internationale Platzhirsch der Vorgehensmodelle. Dies liegt auch daran, dass auf Basis des RUP viele Ab- oder Unterarten (Derivate) entwickelt wurden.

fat models

Vorgehensmodelle wie der RUP oder- das später noch vorgestellte - V-Modell werden intern auch als „fat“ models bezeichnet, weil sie auch auf sehr große Projekte anwendbar sind. Sie lassen sich für die IT von Atomkraftwerken bis zur „Implementierung von Tic-Tac-Toe“ verwenden. Dies bedeutet, dass sehr viele Rollen, Artefakte, Aufgaben, Abläufe und viele Tools definiert bzw. vorgeschlagen werden. Dies geht bis hin zu der Definition von Farben, Dokumenten oder Verzeichnistiefen für Dokumentenablagen - hier ist gemeint, dass es vorgegebene Verzeichnisse mit festgelegten Namen gibt, die eine Verschachtelungstiefe von 9 haben.

Sie unterscheiden sich in diesem Punkt von den agilen Modellen, die sich meist an mittleren und kleineren Projekten orientieren und eben nicht alles definieren und vorschlagen.

2.1 Die 6 wichtigsten RUP Erfahrungen

RUP Erfahrungen

Dem RUP liegen sechs Erfahrungen zugrunde, die im Laufe der Zeit gemacht wurden:

1. **Entwickle die Software iterativ:**

Große Softwaresysteme lassen sich nicht nach einem festen Schema entwickeln. Die Requirements ändern sich im Laufe der Zeit. Architekturen können sich ändern und der spätere Anwender versteht sein Problem zunehmend anders oder besser. Iterationen erlauben es, das Projekt anzupassen und die wichtigsten Risiken besser anzugehen. Jede Iteration soll aber mit einem Release enden, um damit auch besseres Feedback vom späteren Stakeholder einzuholen.

2. **Verwalte die Anforderungen:**

Ein Requirements Management ist zentraler Bestandteil des RUP. Nichts ist schlimmer, als wenn die Wünsche der Anwender nicht erfasst werden und ständig (sich z. B. widersprechende) Änderungen auf Zuruf vieler Personen gemacht werden.

3. **Verwende komponentenbasierte Architekturen:**

In einer Zeit, in der sich Hochsprachen wie C# 3.0 oder Java Mustang schon etabliert haben, mag diese Anforderung antiquiert wirken. In vielen Projekten ist jedoch der Sinn von Komponenten nicht klar oder wird nicht ausgenutzt. Moderne Architekturen - wie sie fast nur von modernen OO-Sprachen realisiert werden können - sind ein Teilgarant für erfolgreichere Projekte. Aus diesem Grund macht der RUP sogar teilweise Vorgaben für die Architektur des Systems. Eine Wiederverwendbarkeit von Komponenten - wie sie CORBA, COM, .NET oder J2EE ermöglichen, soll angestrebt werden.

4. **Modelliere Software visuell:**

Eine visuelle Abstraktion vom eigentlichen Programmcode hilft, den Überblick zu behalten. Interaktionen werden sichtbar und die Komplexität des Systems kann besser verborgen werden. Hier soll natürlich wenn möglich der „Industriestandard“ UML verwendet werden.

5. **Überprüfe die Softwarequalität kontinuierlich:**

Das Fehlen eines Qualitätskonzeptes ist nach RUP ein recht häufiger Grund für das Scheitern von Projekten. Qualitätskontrollen sind daher integraler Bestandteil fast aller Phasen des RUP, das von jeder Person wahrgenommen wird. Dazu gehören ggf. auch Maßzahlen, die die Qualität der Software oder Tests wiedergeben.

6. **Verwalte Softwareänderungen:**

Veränderungen der Software sind unausweichlich und essenziell. Daher muss effektiv kontrolliert und protokolliert werden. Dazu gehören die drei Gruppen:

- Configuration Management
 - Change Request Management und
 - Status Measurement Management.
- (Siehe dazu auch Lerneinheit SVN - Versionskontrolle)

2.2 RUP Tools

Sie sollten sich eine DVD oder einen Download des RUP zur Evaluation besorgen, um einen Überblick zu den Funktionen und den zur Verfügung stehenden Werkzeugen zu erhalten.

www.ibm.com/software/awdtools/rup.

Die Abbildung zeigt die wichtigsten Werkzeuge des RUP.

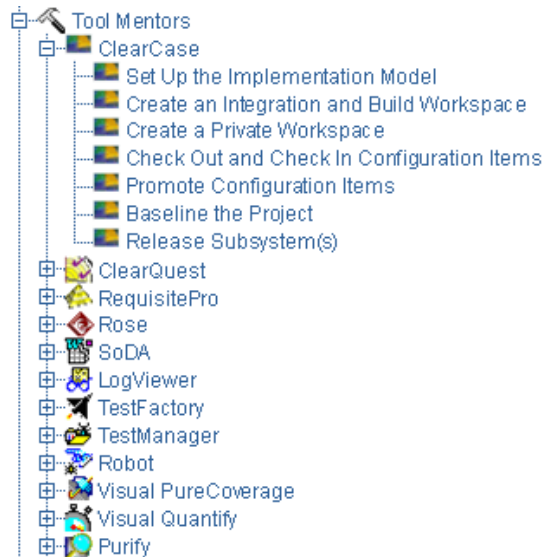


Abb.: RUP Tools

Kurze Erläuterung einiger Werkzeuge:

- **Clear Case**® ist ein Werkzeug für das Konfigurationsmanagement
- **Clear Quest**® dient dem Change Management
- **Requisite Pro**® ist ein Tool für das Management von Requirements
- **Rose**® ist ein bekanntes Werkzeug zur UML Modellierung
- **Robot**® erlaubt, funktionale Tests zu managen / automatisieren - es adressiert funktionale Tests, Regressionstests, „smoke tests“ und Viele mehr

Diese Werkzeuge sollen die genannten „Best Practices“ geeignet unterstützen:

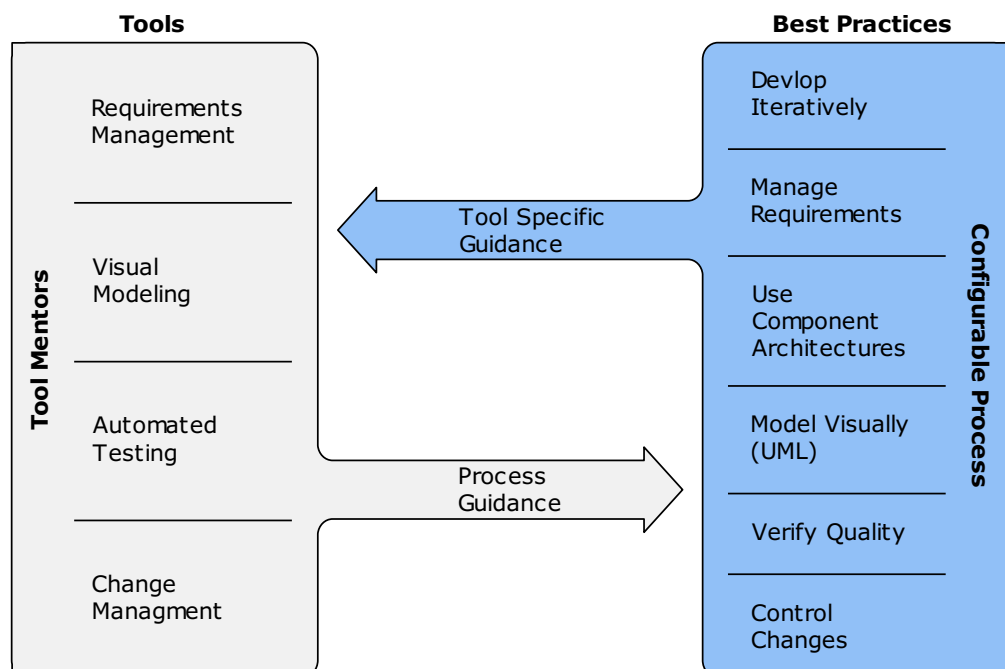


Abb.: Die RUP Tools

unterstützen die „best practices“

21.02.2018

2.3 RUP Tools

Wie in der Abbildung erkennbar, wird dem Projektmanager eine web-basierte Oberfläche zur Verfügung gestellt, in der sowohl die Dokumente, die Workflows als auch die Werkzeuge erreichbar sind. Dies ermöglicht ein effizientes Projektmanagement.

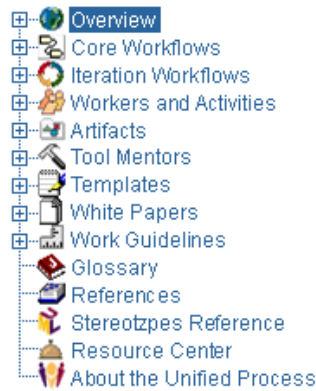


Abb.: Die integrierte Umgebung des RUP

Abschließend die Ansicht von Requisite Pro, welches die Dokumente / Requirements verwaltet, die im Projekt erfasst werden.

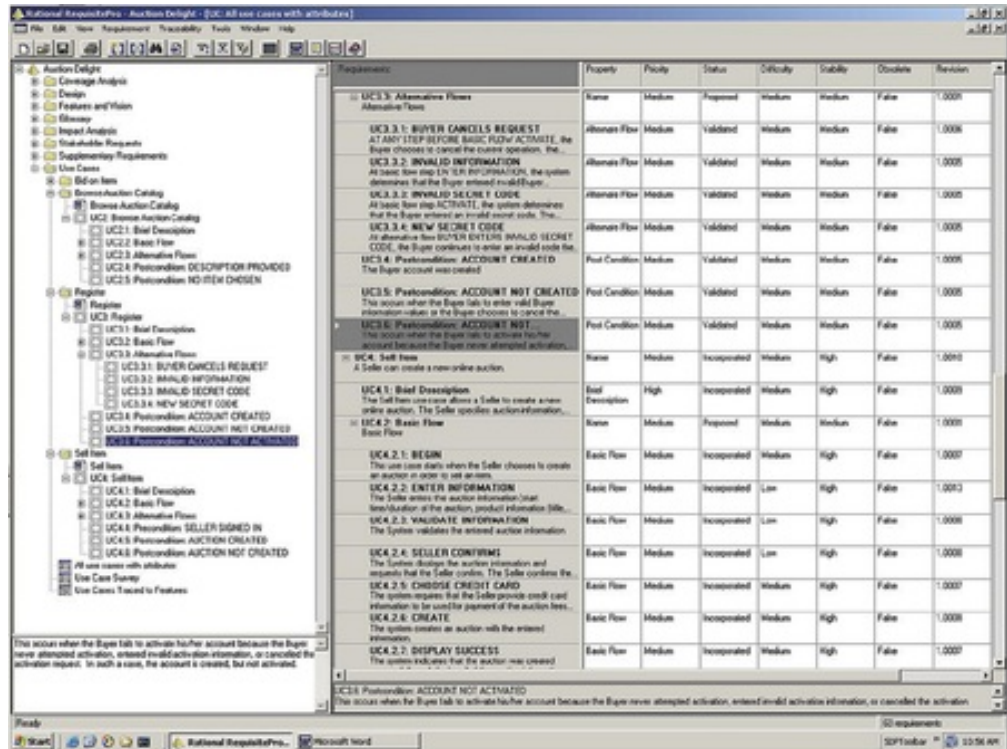


Abb.: Requisite Pro

2.4 Der RUP Lifecycle

Dem RUP liegt eine interaktive / **zyklische Vorgehensweise** zugrunde, die aus den nachfolgenden Bestandteilen besteht:

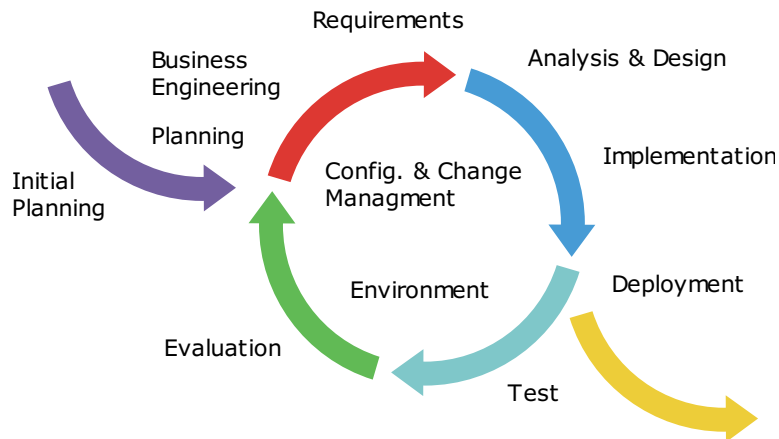


Abb.: „RUP cycle“

Die in diesem „cycle“ genannten globalen Aktivitäten werden in der wohl bekanntesten Abbildung des RUP in Phasen eingeteilt. Die farbigen Flächen in den Koordinaten zeigen, wie intensiv die Aktivitäten in der jeweiligen Phase durchgeführt werden.

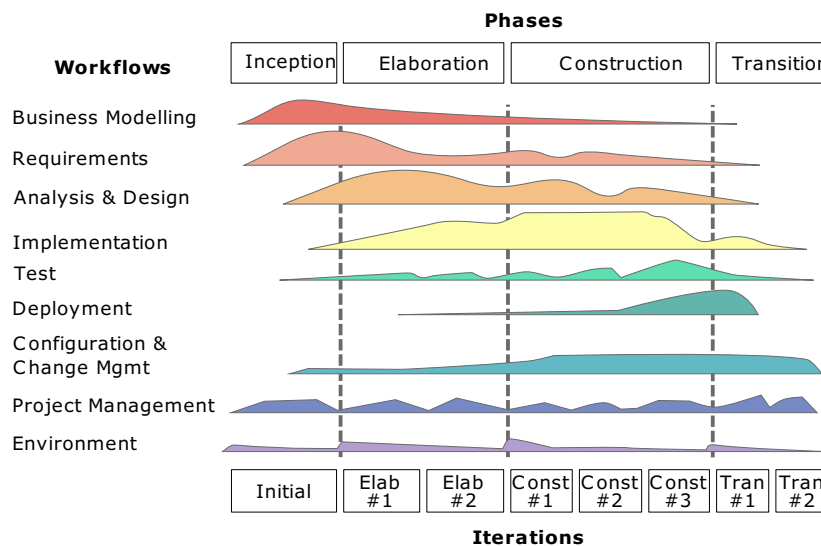


Abb.: Intensität der globalen Aktivitäten in den RUP Phasen

Beim RUP hervorzuheben sind die vier Phasen, die bitte **nicht** mit Analyse, Design, Implementierung und Übergabe gleichzusetzen sind:

- **Inception:** In dieser ersten Phase werden die grundlegenden Anforderungen (Business Requirements) gelegt. Weiterhin grundlegende Erfolgsfaktoren wie Marktanalysen, Finanz-, Income-Rechnungen getätigt, Use-Cases (siehe Lerneinheit Unified Modeling Language) ermittelt, Risiken analysiert und Projektplanungen erstellt.
- **Elaboration:** Hier nimmt das Projekt Form an. Die ersten Architekturen werden sichtbar. Die meisten Use-Cases sind fertig. Ein Projektplan für das gesamte Projekt liegt vor. Ein erster Prototyp, der die grundlegende Architektur (Durchstich) zeigt, ist erstellt und lauffähig. Projektrisiken und Anforderungen werden erneut überprüft.
- **Construction:** Schwerpunkt ist hier die Entwicklung. Hier können mögliche Probleme ebenfalls mit vielen Iterationen adressiert werden.
- **Transition:** In Endphase des Projektes verstärkt sich der Einsatz beim Endbenutzer. In dieser Phase wird die Software übergeben und vom Endbenutzer geprüft.

Die folgende Abbildung zeigt, dass alle vier Phasen wiederum von beliebig vielen Iterationen durchsetzt sein können:

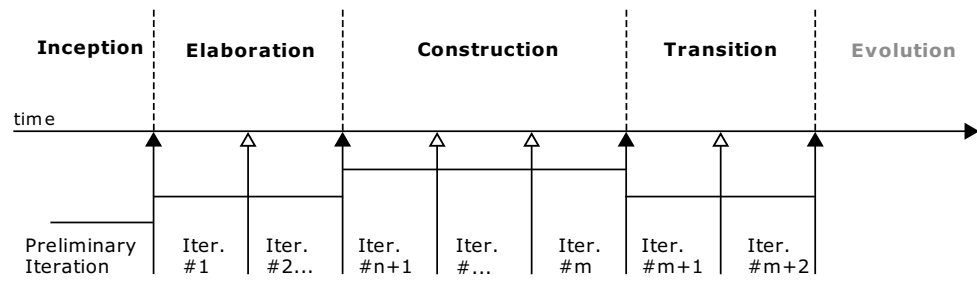


Abb.: Iterationen in den 4 Phasen

2.5 RUP Rollen / Personen

Der RUP sieht für größere Projekte eine Reihe von Rollen vor.

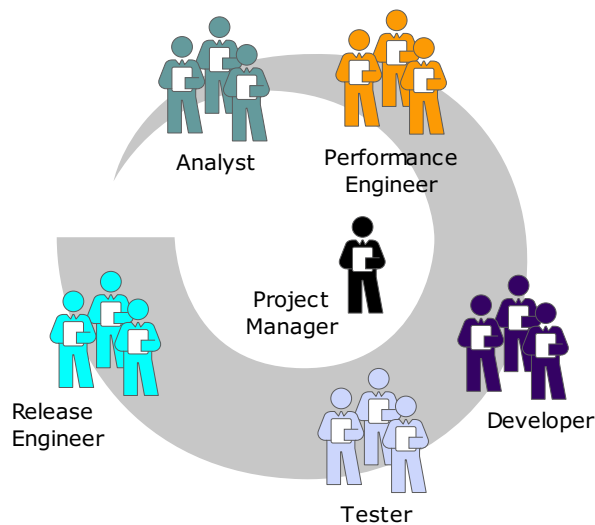


Abb.: Mögliche
Projektrollen

Beim RUP werden in den frühen Phasen Rollen wie „Analyst“ und „Performance Engineer“ (<http://www.ibm.com>) benötigt. Im Zentrum des ganzen Projektes stehen der Projektmanager, der Developer und der Tester. Release Engineers stehen sowohl im ganzen Prozess, als auch besonders am Ende des Prozesses und sind verantwortlich für die Übersetzung des Codes und für das Zusammenstellen und die Auslieferung der Quellen in das Zielsystem. Die Rolle beinhaltet viel vom Build- und Versionsmanagement welches in einer späteren Lerneinheit noch behandelt wird.

Die folgende Abbildung mag für den Leser ungewöhnlich detailliert erscheinen, ist aber für **Großprojekte** nicht ungewöhnlich. Sie zeigt mögliche Rollen im Projekt für die verschiedenen Bereiche wie Architektur, Development, Management usw.

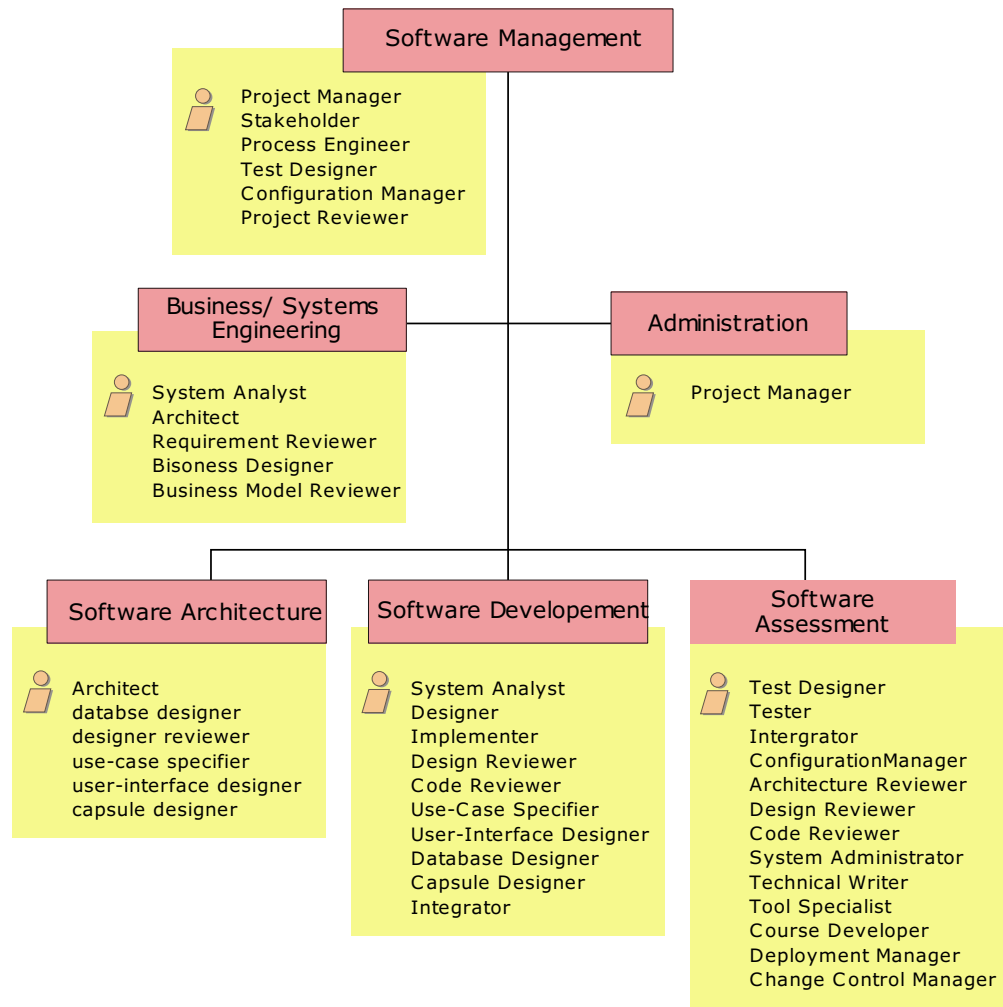


Abb.: Aufgabenbereiche in Großprojekten

2.6 Rollen, Aufgaben und Artefakte

Das Beispiel zeigt einen extrem kleinen Ausschnitt aus den möglichen Abläufen im RUP. In diesem Ablauf sind Personen mit Rollen (3 rosa Figuren), Aufgaben (spitzer Kasten) und auch Artefakte bzw. Arbeitsergebnisse dargestellt.

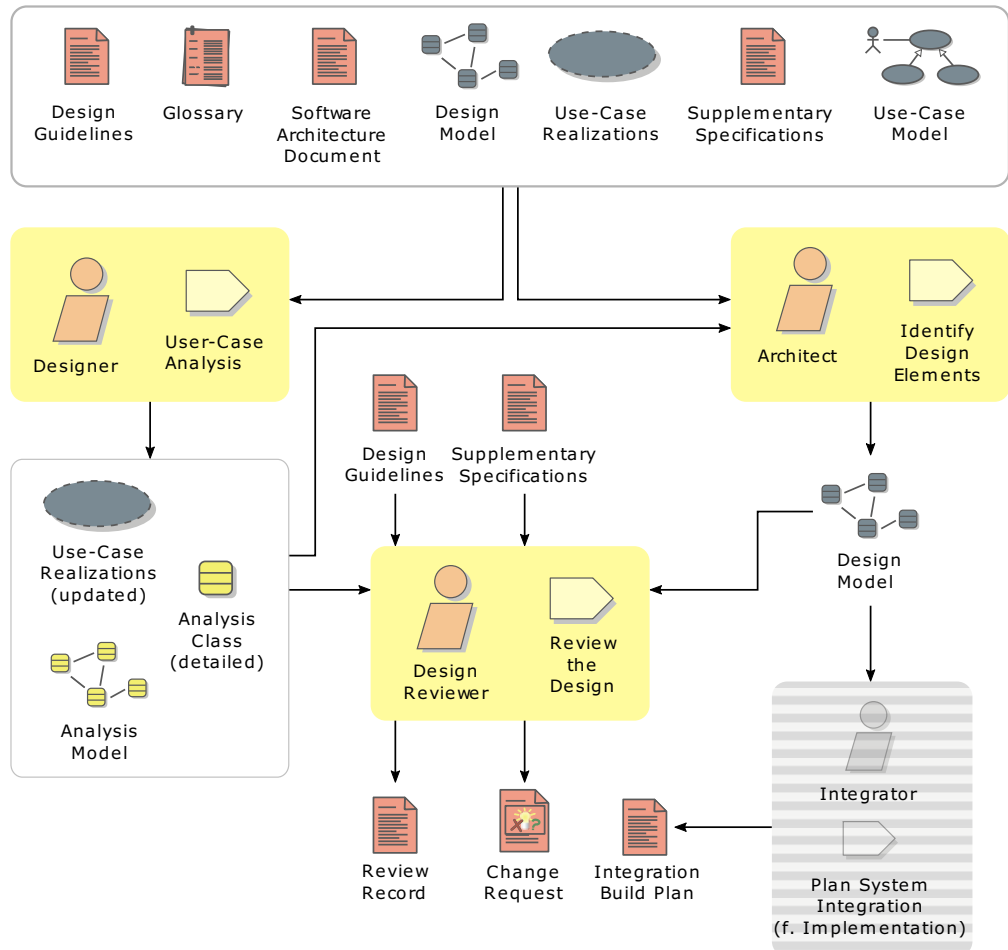


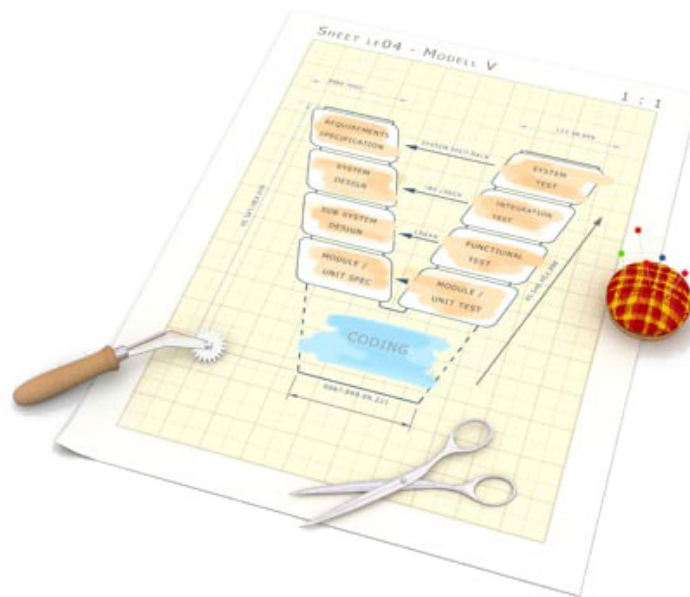
Abb.: Ausschnitt aus der Elaboration-Phase

Die Arbeit des Design Reviewers

Betrachten wir einmal kurz den **Design Reviewer**. Seine Arbeit beginnt, wenn der Designer ein Design fertiggestellt hat. Er erhält als „Input“ die Use-Cases und Klassen, ein Analysemodell und vom Architekten ein Designmodell. Der Arbeit des Design Reviewers liegen weiterhin - global - die Design Guidelines und ergänzende Spezifikationen zugrunde. Die Arbeitsergebnisse sind hauptsächlich **Change Requests**, die Vorschläge zu Änderungen am Design enthalten. Seinen Arbeitsschritt fasst er in einem **Review Record** protokollarisch zusammen.

An diesem Beispiel ist erkennbar, dass ein solches Modell in Großprojekten extrem viel Erfahrung und Tailoring bedeutet. Designer und Architect müssen dabei exakt wissen, was sie zu tun haben, wie sie zusammenarbeiten, sich **abstimmen** und **abgrenzen**. Der Input des Design Reviewers - Klassen vom Designer und das Designmodell vom Architekten - muss **konsistent** sein.

3 Das V-Modell



Bundesministerium
1996

Das V-Modell wurde ursprünglich im Auftrag des Bundesministeriums für Verteidigung von der Industrieanlagen-Betriebsgesellschaft mbH (IABG) erstellt. Es gilt seit Sommer 1996 als verbindlich einzusetzende Vorschrift für Projekte im Bereich der Bundesverwaltung des Bundesinnenministeriums. Das V-Modell ist nur für Deutschland relevant und wird der Vollständigkeit halber hier behandelt.

Durch jährliche **Änderungskonferenzen** von Industrie- und Behördenvertretern ist der Einfluss der Anwender beim kontinuierlichen Pflege- und Änderungsprozess des V-Modells sichergestellt. Im Gegensatz zu anderen Prozessmodellen wurde das V-Modell unter mehreren Anwendungsaspekten entwickelt und ist als Vertragsgrundlage, als Arbeitsanleitung und als Kommunikationsbasis geeignet.

Aufgrund seines organisationsneutralen Regelungsteils wird das V-Modell heute teilweise auch als **Entwicklungsstandard** in der Industrie verwendet. Die Einführung des V-Modells in die Industrie wurde dadurch wesentlich begünstigt, so dass es keinen Nutzungsrechten unterliegt, d. h. es darf beliebig kopiert und eingesetzt werden. [DHM98]

Öffentliche Auftraggeber


Verbreitet ist das V-Modell jedoch bei Behörden und gerade bei öffentlichen Aufträgen, da der Bund im V-Modell viele Sicherungen zur Hand hat, die einen frühen Missbrauch oder eine Fehlentwicklung des Projektes unterbinden können.

Es soll an dieser Stelle aber auch nicht verschwiegen werden, dass zumindest dem „alten“ V-Modell teilweise auch ein **zweifelhafter Ruf** anhaftet. Mitarbeiter von beauftragten Firmen klagen häufig über starre Vorgehensweisen und über eine Dokumentenzentriertheit, die vielleicht beim Umgang mit öffentlichen Geldern notwendig ist, aber den Mitarbeitern das Leben dennoch ziemlich schwer macht. Dieser Ruf kann allerdings auch darin begründet sein, dass mangelnde Erfahrung im Tailoring des Modells vorliegt.

Anders kann dies beim V-Modell XT sein, welches 2005 verabschiedet wurde. Mehr Informationen finden Sie auf der Homepage des V-Modell XT.

<http://www.v-modell.iabg.de>

3.1 Struktur

Das V-Modell umfasst ein dreistufiges Standardisierungskonzept  [Vmod98b] , welches in untenstehender Abbildung bildhaft dargestellt ist:

	Projekt- management	Software- erstellung	Qualitäts- sicherung	Konfigurations- management
Vorgehen				
Methoden				
Werkzeuge				

Abb.: Struktur des V-Modells

Die Ebenen Vorgehen, Methoden und Werkzeuge haben dabei folgende Bedeutung:

- **Das Vorgehensmodell (Wann ist etwas zu tun?):**
Diese Ebene legt fest, welche Tätigkeiten im Verlauf der Systementwicklung durchzuführen und welche Ergebnisse dabei zu produzieren sind. Außerdem bestimmt die Ebene, welche Inhalte diese Ergebnisse haben müssen.
- **Die Methodenzuordnung (Wie ist etwas zu tun?):**
Hier erfolgt die Bestimmung, mit welchen Methoden die auf der ersten Ebene festgelegten Tätigkeiten durchzuführen und welche Darstellungsmittel in den Ergebnissen zu verwenden sind.
- **Die funktionalen Werkzeuganforderungen (Womit ist etwas zu tun?):**
Auf dieser Ebene ist fixiert, welche funktionalen Eigenschaften die Werkzeuge aufweisen müssen, die bei der Systementwicklung eingesetzt werden sollen.

Werkzeuge berücksichtigen

Der Kern des Standards ist die Beschreibung des Software-Entwicklungsprozesses als Vorgehensmodell. Dabei werden in den Begriff V-Modell die Teile Methodenzuordnung und funktionale Werkzeuganforderungen mit eingeschlossen, weil diese als Ergänzung zum Vorgehensstandard zu sehen sind.

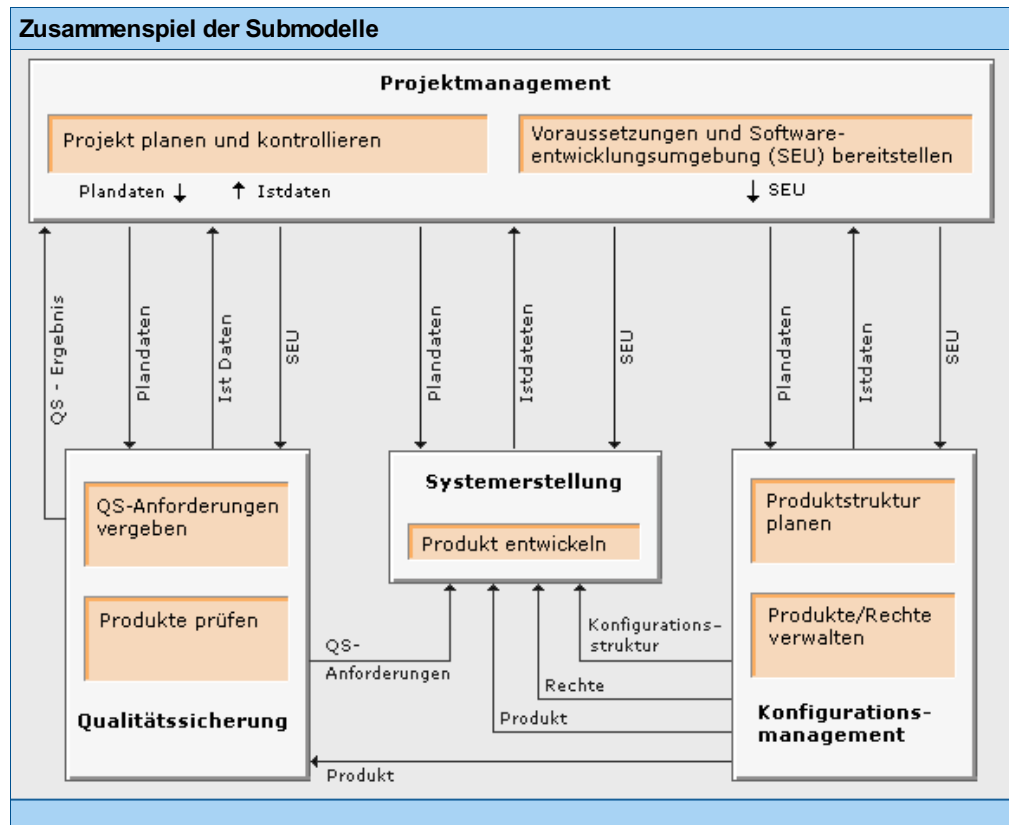
3.2 Vorgehensmodell: Submodelle

Das V-Modell enthält Rahmenregelungen für die Software-Entwicklung und betrachtet dabei den Gesamtprozess, der in Tätigkeitsbereiche - Systemerstellung, Projektmanagement, Konfigurationsmanagement und Qualitätssicherung - untergliedert ist. Diese Tätigkeitsbereiche werden in Form von Submodellen beschrieben.

 [Vmod98b]



Rolloverbild



Textversion: Zusammenspiel der Submodelle

Die **Systemerstellung** (SE) erstellt das System bzw. die Softwareeinheit.

Das **Projektmanagement**(PM) plant, initiiert und kontrolliert den Prozess und informiert die Ausführenden der übrigen Submodelle.

Die **Qualitätssicherung**(QS) gibt Qualitätsanforderungen, Prüffälle und Kriterien vor und unterstützt die Produkte bzw. den Prozess hinsichtlich der Einhaltung von Qualitätsanforderungen und Standards.

Das **Konfigurationsmanagement**(KM) verwaltet die Produkte. Es stellt sicher, dass die Produkte eindeutig identifizierbar sind und Produktänderungen nur kontrolliert durchgeführt werden. Die vier Submodelle sind eng miteinander vernetzt und beeinflussen sich über den Austausch von Produkten und Ergebnissen gegenseitig. Das Zusammenspiel auf höchster Ebene ist in der Abbildung dargestellt.

3.3 Vorgehensmodell: Aktivitäten- und Produktfluss

Im V-Modell wird der Entwicklungsprozess als eine Folge von Tätigkeiten (Aktivitäten) und deren Ergebnissen (Produkten) beschrieben.

Aktivitäten werden in diesem Zusammenhang als Tätigkeiten im Rahmen des System-Entwicklungsprozesses definiert, die hinsichtlich ihres Ergebnisses und ihrer Durchführung genau beschrieben werden können. Eine Aktivität kann beispielsweise die inhaltliche Änderung eines Produktes zum Gegenstand haben.

Ein **Produkt** stellt abstrakt eine Information dar, die zur Durchführung einer Aktivität benötigt wird oder die nach Durchführung einer Aktivität vorliegt.

Die nächste Abbildung gibt beispielhaft einen Überblick über die Abfolge von Aktivitäten und Produkten zur Erstellung eines Projekthandbuches.

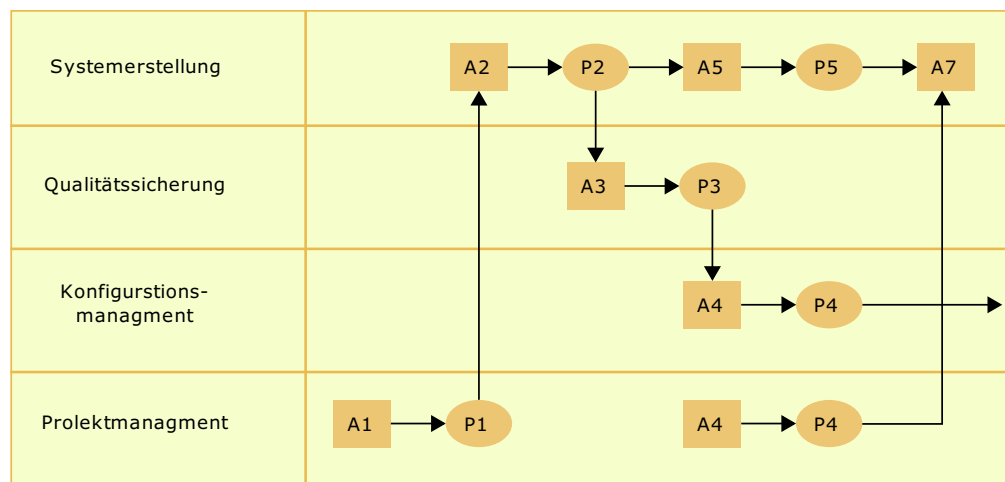


Abb.: Beispiel für den Aktivitäten- und Produktfluss über die vier Submodelle

Zu jeder Aktivität existiert eine Beschreibung die als **Arbeitsanleitung** dient und der ein Projektfluss zugrunde liegt. Im Projektfluss wird angegeben:

- welche Produkte als Eingangsprodukte benötigt werden
- wo sie zuletzt bearbeitet wurden
- welche Produkte erzeugt oder modifiziert werden
- in welcher Folgeaktivität die erzeugten / modifizierten Produkte verwendet werden

3.4 Vorgehensmodell: Submodell Systemerstellung (SE)

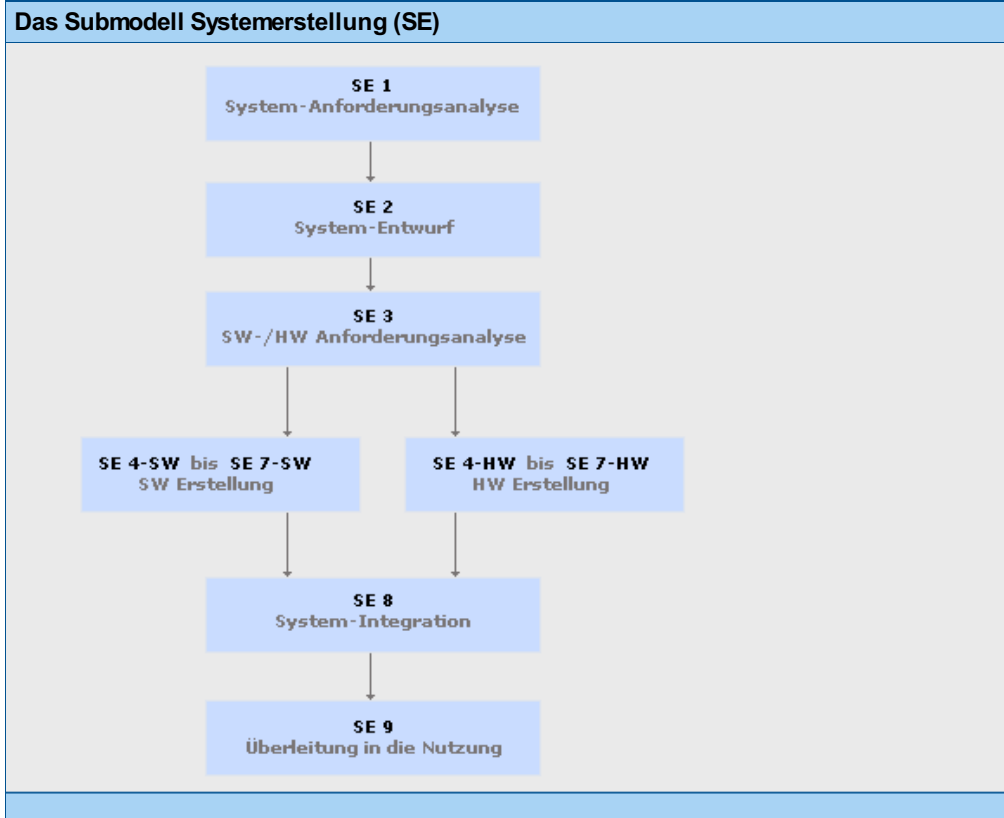
Jedes Submodell besteht aus Hauptaktivitäten, die wiederum in Teilaktivitäten gegliedert werden können. Das wichtigste Submodell des V-Modells ist die Systemerstellung. Es umfasst alle der Systemerstellung dienenden Aktivitäten und die jeweiligen Entwicklungsdokumente.

Die Animation zeigt die Hauptaktivitäten und deren Ablauf im Submodell Systemerstellung. (aus

 [Vmod98b])



Rolloverbild



Textversion: Submodell Systemerstellung (SE)

Die **Hauptaktivitäten** im Submodell Systemerstellung (SE) sind:

(SE 1) System-Anforderungsanalyse

- Beschreibung der Anforderungen an das zu erstellende System und seine technische und organisatorische Umgebung;
- Durchführung einer Bedrohungs- und Risikoanalyse;
- Erarbeiten eines fachlichen Modells für Funktionen, Daten und Objekte.

(SE 2) System-Entwurf

- Zerlegung des Systems in Segmente sowie in SW (Software)- und HW (Hardware)-Einheiten.

(SE 3) Software-/Hardware-Anforderungsanalyse

- Die technischen Anforderungen an die SW- und gegebenenfalls HW-Einheiten werden präzisiert. Von hier ab spaltet sich der weitere Fortgang in die SW-Entwicklung und gegebenenfalls in die HW-Entwicklung.

(SE 4) Software-/Hardware-Grobentwurf

(SE 5) Software-/Hardware-Feinentwurf

(SE 6) Software-/Hardware-Implementierung

(SE 7) Software-/Hardware-Integration

(SE 8) System-Integration

- Integration der verschiedenen Software- und Hardwareeinheiten zu einem Segment und Integration der Segmente (falls vorhanden) zum System

(SE 9) Überleitung in die Nutzung

- Beschreibung aller Tätigkeiten, die notwendig sind, um ein fertig gestelltes System an der vorgesehenen Einsatzstelle zu installieren und in Betrieb zu nehmen.

3.5 Entwicklungsdokumente

Mit der Anwendung des V-Modells wird sichergestellt, dass für die Software-Entwicklung alle sinnvollen und erforderlichen Entwicklungsdokumente als Produkt- und Dokumentenmuster vorgegeben werden. Die nachstehende Abbildung zeigt einen Überblick der Entwicklungsdokumente des Vorgehensmodells.

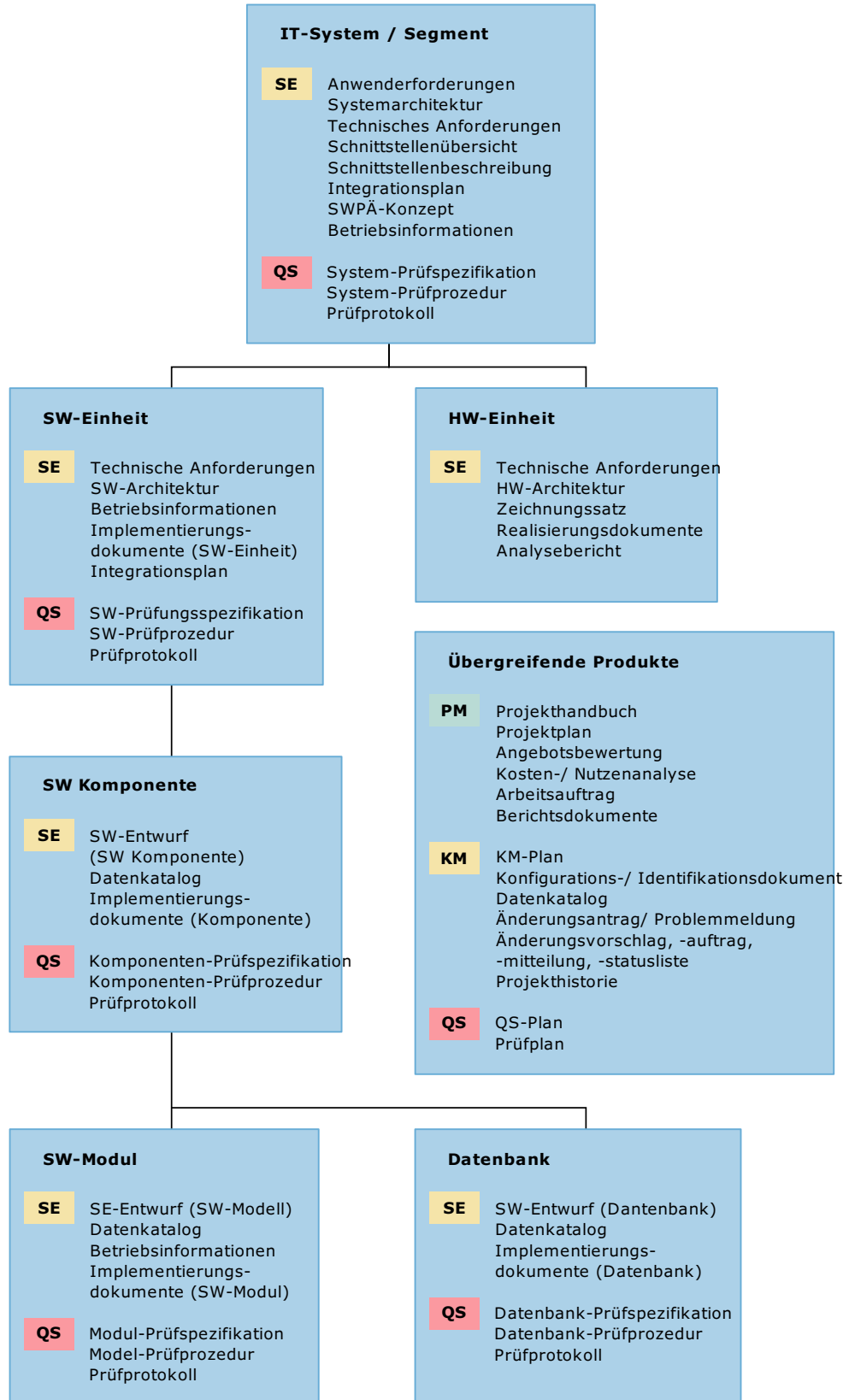


Abb.: Übersicht zu den Entwicklungsdokumenten des V-Modells

3.6 Tailoring im V-Modell

Das Vorgehensmodell zeichnet sich durch Allgemeingültigkeit sowie durch Firmen- und Projektunabhängigkeit aus. Um es für ein konkretes Projekt einzusetzen, kann individuell entschieden werden, welche Aktivitäten und Entwicklungsdokumente für das Projekt aus sachlichen Gründen erforderlich sind. Übermäßige **Papierflut**, sinnlose Dokumente, aber auch das Fehlen wichtiger Dokumente sollte in jedem Fall verhindert werden. Diese projektspezifische Anpassung wird Tailoring genannt.



Ein wesentliches Ergebnis der Anpassung des V-Modells an die projektspezifischen Bedingungen ist das erstellte **Projekthandbuch**. Es definiert den zu erbringenden Leistungsumfang und bildet die Handlungsgrundlage für alle Projektvorbereitungen.

3.7 Methodenzuordnung

Die Methodenzuordnung regelt im Zusammenwirken mit dem Vorgehensmodell den Einsatz von Methoden bei der Systemerstellung (SE) und den begleitenden Tätigkeiten für die Qualitätssicherung (QS), das Konfigurationsmanagement (KM) und das technische Projektmanagement (PM).

Durch die Methodenzuordnung wird der Einsatz von Methoden in allgemein gültiger Weise geregelt. Diese Allgemeingültigkeit wird erreicht, durch die Beschränkung auf elementare Methoden bei der Festlegung der Methodenzuordnung.

Als Elementarmethoden werden methodische Vorgehensweisen bezeichnet, die eine spezifische, abgegrenzte Sicht des Systems (z. B. funktionale, datenorientierte, objektorientierte Sicht) und/oder einen bestimmten Ausschnitt der Systementwicklung (z. B. Analyse, Grobentwurf) oder der begleitenden Tätigkeiten (PM, KM, QS) beschreiben. [Vmod98b]

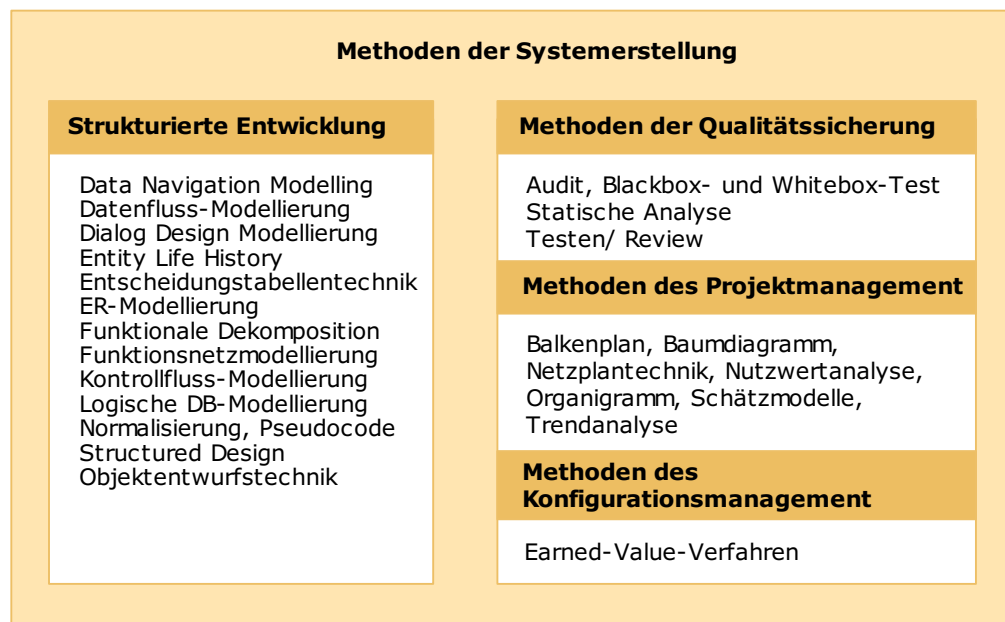



Abb.: Überblick der Elementarmethoden für das Submodell Systemerstellung

Als komplexe Methoden werden Vorgehensweisen bezeichnet, die verschiedene methodische Komponenten beinhalten und diese in einer Gesamtmethode integrieren, z. B. die Methode Structured Analysis - siehe Abbildung.

3.8 Zuordnungstabellen

Die Methodenzuordnung erfolgt in Verbindung mit dem Vorgehensmodell, d. h. für jede im Projekt durchzuführende Tätigkeit ist eine geeignete Methode zu finden. Dafür werden so genannte Zuordnungstabellen verwendet. Die Tabelle zeigt beispielhaft die für die Aktivität (SE1.2) geeigneten Methoden auf.


Teilaktivität SE 1.2: Anwendungssystem beschreiben	
Produkt: Anwenderanforderungen	
Kapitel:	Methoden-Verweis:
3. IT-Sicherheit	
4. Fachliche Anforderungen	
6.1 Grobe Systembeschreibung	Datenflussmodellierung Klassen-/Objektmodellierung Zustandsmodellierung usw.

Tab.: Beispiel einer Methodenzuordnung
(aus  [Vmod98b] S.2))

Aktivitäten-Methoden-Zuordnung

Die folgende Tabelle enthält eine Aktivitäten-Methoden-Zuordnung für die Hauptaktivität Systemanforderungsanalyse (SE1). In der Methodenzuordnung ist die Eignung einer Methode für eine bestimmte Aktivität durch Kennzeichen geregelt. So soll beispielsweise ein * darauf hinweisen, dass eine Methode nur dann geeignet ist, wenn bestimmte Einsatzbedingungen vorliegen.

	SE-Aktivitäten	Geeignete Methoden
SE 1	Systemanforderungsanalyse	
SE 1.1	Ist-Aufnahme/-Analyse durchführen	ER-Modellierung* Datenfluss-Modellierung Funktionale Dekomposition Objektorientiert: Klassen-/Objektmodellierung* Use-Case-Modellierung* Zustandsmodellierung* Interaktionsmodellierung* Prozessdiagramme*
SE 1.2	Anwendungssystem beschreiben	Datenfluss-Modellierung Objektorientiert: Class-Responsibility-Collaboration* Klassen-/Objektmodellierung* Subsystemmodellierung* Prozessdiagramme* Use-Case-Modellierung* Zustandsmodellierung* Interaktionsmodellierung*
SE 1.3	Kritikalität und Anforderungen an die Qualität definieren	Zuverlässigkeitsmodelle* Failure Mode Effect Analysis*
SE 1.4	Randbedingungen definieren	Klassen-/Objektmodellierung* Class-Responsibility-Collaboration* Prozessdiagramme* Use-Case-Modellierung* Zustandsmodellierung* Interaktionsmodellierung*
SE 1.5	System fachlich strukturieren	Systemverhaltensmodelle* Funktionsnetzmodellierung* ER-Modellierung* Datenfluss-Modellierung Funktionale Dekomposition Expertisemodell* Objektorientiert: Subsystemmodellierung* Use-Case-Modellierung* Klassen-/Objektmodellierung* Class-Responsibility-Collaboration* Zustandsmodellierung* Interaktionsmodellierung*
SE 1.6	Bedrohung und Risiko analysieren	keine Zuordnung
SE 1.7	Forderungscontrolling durchführen	keine Zuordnung
SE 1.8	Software-Pflege- und Änderungs-Konzept erstellen	Organigramm Balkenplan

Tab.: Aktivitäten- Methoden- Zuordnung für die Systemanforderungsanalyse
(Auszug aus  [Vmod98b])

3.9 Schnittstellen

Beim Einsatz mehrerer Methoden ist zu beachten, dass es häufig Schnittstellen zwischen den Methoden gibt. In nachstehender Abbildung sind die Schnittstellen zwischen den strukturierten Methoden dargestellt.

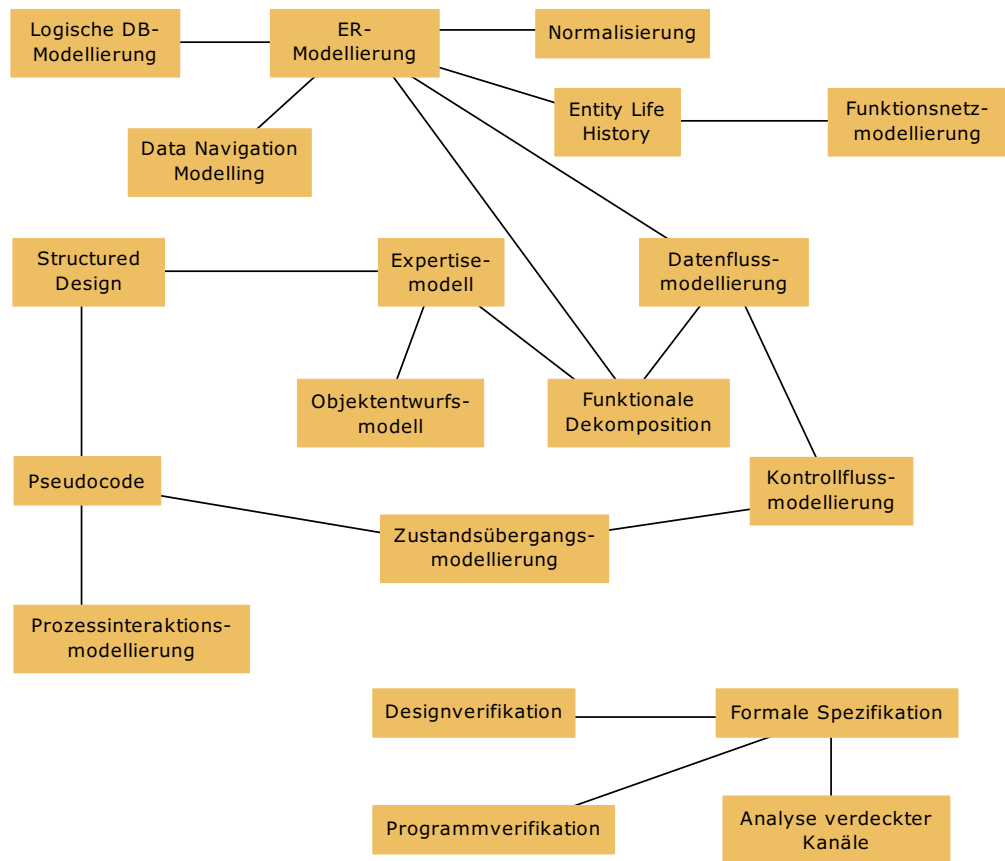


Abb.: Schnittstellen
strukturierter Methoden
[Vmod98b]

3.10 Werkzeuganforderungen

Nachdem in der Methodenzuordnung definiert wurde, wie die jeweiligen Aktivitäten auszuführen sind und welche Darstellungsmittel in den Ergebnissen zu verwenden sind, wird auf der dritten Ebene festgelegt, welche funktionalen Eigenschaften die Tools aufweisen müssen, die für die Realisierung der Aktivitäten eingesetzt werden sollen. Es wird in diesem Sinne ein **Werkzeugstandard** definiert.

Mit Hilfe des Werkzeugstandards wird festgelegt, gegen welche funktionalen Kriterien Werkzeuge zu überprüfen sind, wenn sie in einer SEU zum Einsatz kommen sollen. Der Werkzeugstandard enthält jedoch **keine** technisch-organisatorischen Anforderungen, wie z. B. die Vorgabe einer Rechnerarchitektur, personelle Infrastruktur etc., da solche Gesichtspunkte sehr stark vom jeweiligen Umfeld abhängig sind. Da für eine Werkzeugauswahl sowohl funktionale als auch technische und organisatorische Anforderungen betrachtet werden müssen, ist es im konkreten Anwendungsfall notwendig, neben den funktionalen Anforderungen auch die Einsatzbedingungen zu analysieren.

 [Vmod98b]

In der Abbildung wird die Vorgehensweise zur **Werkzeugauswahl** veranschaulicht, bei der alle relevanten Aspekte berücksichtigt werden.

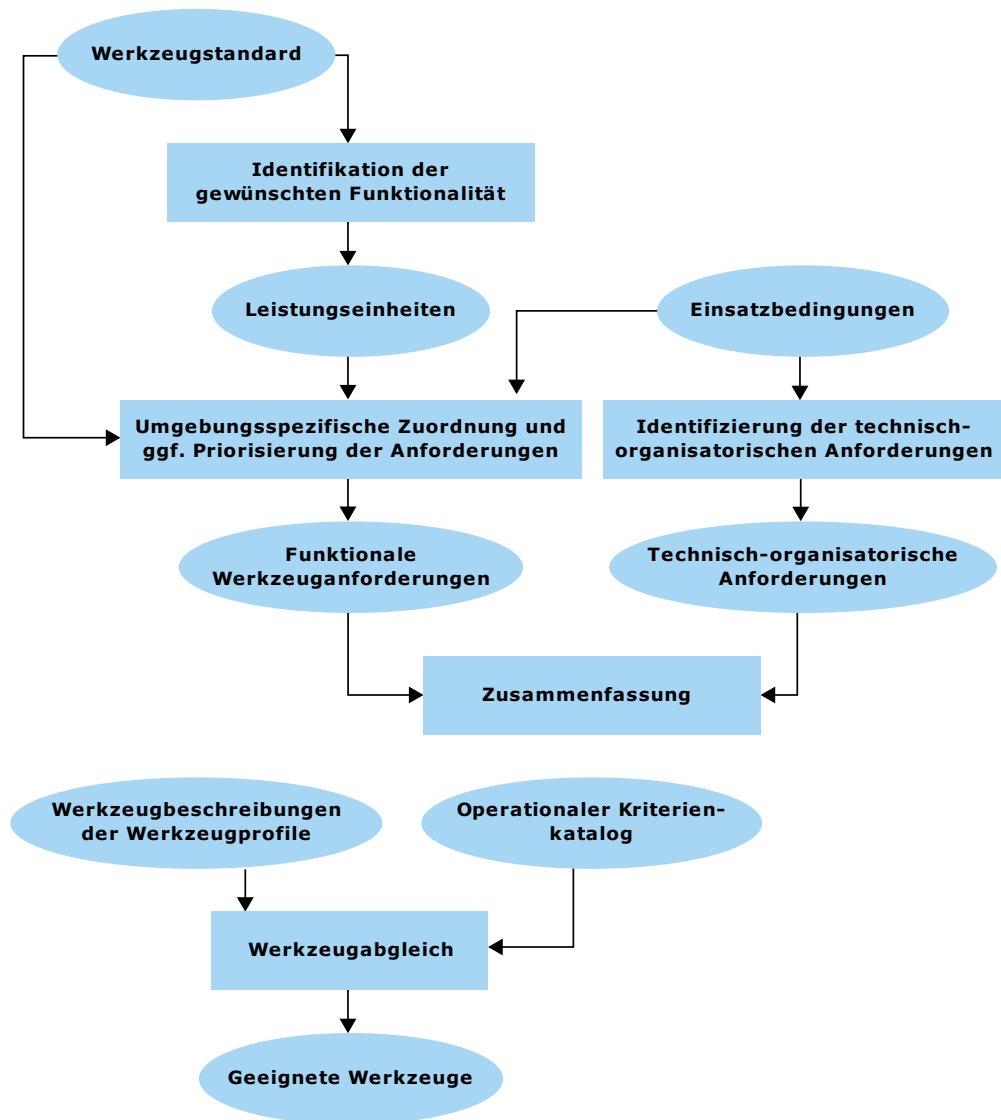




Abb.: Vorgehensweise bei der Auswahl von Werkzeugen (in Anlehnung an

 [VMod98])

3.11 Zusammenfassung

Das V-Modell ist gekennzeichnet durch einen hohen generischen Charakter. Die Ursache liegt in den vielfältigen Einsatzschwerpunkten für das V-Modell. So dient es u. a. als Vertragsgrundlage, als Kommunikationsbasis zwischen den Beteiligten am Projekt und als Arbeitsanleitung. Die Anpassung des vorgegebenen generischen Modells an die spezifischen Bedingungen eines konkreten Anwendungsfalls (Tailoring) ist in der Regel mit **erheblichem Arbeitsaufwand** verbunden.

- Das V-Modell kommt vorrangig bei komplexen Projekten zur Anwendung und bietet dann den Vorteil, dass die Tätigkeiten Projektmanagement, Qualitätssicherung und Konfigurationsmanagement in den Gesamtprozess integriert werden.
- Für IT-Projekte mit geringem Umfang ist die Anwendung des V-Modells weniger zu empfehlen.

Seit der Einführung des V-Modells wurden von verschiedenen Herstellern, Werkzeuge für das Prozessmanagement entwickelt, die das V-Modell umsetzen und den - im Einzelfall - aufwändigen Prozess des Tailoring unterstützen. Ein Beispiel ist das Softwareprodukt in-Step der Firma  microTool. Informationen zu diesem Prozessmanagement-Tool können von der Webseite heruntergeladen werden. Dabei wird aufgezeigt, wie mit Hilfe von in-Step das V-Modell umgesetzt werden kann.

4 OEP

Hinter dem oose Engineering Process (OEP) steht maßgeblich Bernd Oesterreich, der mit seinen Werken zu UML und Analyse-Design viel Vorarbeit geleistet hat. Der OEP ist in Teilen dem RUP ähnlich, wird in vielen Unternehmen eingesetzt und natürlich auch von Oesterreichs eigenen Firma entsprechend gefördert.


Als Literatur wird daher auch das Buch von OESTEREICH ET. AL empfohlen:

BERND OESTEREICH, CLAUDIA SCHRÖDER, MARKUS KLINK, GUIDO ZOCKOLL (2006)

OEP - oose Engineering Process: Vorgehensleitfaden für agile Softwareprojekte

Dpunkt Verlag, ISBN-13: 978-3898644075

Eine gute Übersicht bieten auch die entsprechenden Webseiten des OEP

 <http://www.oose.de/oep>

Schauen Sie sich die Webseite genauer an und arbeiten Sie für sich die Merkmale des OEP heraus.



Hinweis

OEP ist ein konkreter Vorgehens- / Prozessleitfaden für die objektorientierte Softwareentwicklung! Es ist kein universelles Vorgehensmodell.

5 Extreme Programming (XP)



KENT BECK



Hinweis

Es gibt wohl kaum ein Vorgehensmodell, dass mehr für Furore gesorgt hat als XP.

Anfänglich scharf kritisiert, gibt es mittlerweile eine große weltweite **XP Community**. Viele Ideen von XP sind mittlerweile auch in großen Projekten etabliert.

Zu Recht kann man sagen, dass XP die Denkweise und den Hype des agilen Vorgehens angestoßen hat und ihm mit seinen „**extremen**“ **Ideen** zu einer breiten Popularität verholfen hat. **KENT BECK** wollte bewusst mit extremen Ideen provozieren, um einen **Gegenpol** zu alten unbeweglichen Modellen zu bilden.

Kurzbeschreibung des Extreme Programming XP

XP ist eine Vorgehensweise für die Entwicklung von Software in Projekten. Es beinhaltet praxisnahe Anleitungen für Entwickler und Projektleiter, die besonders **Werte** und die soziale Interaktion des Menschen in den Vordergrund stellen. Einige Denkanstöße sind insbesondere aus der Sicht des letzten Jahrhunderts recht „extrem“. XP versteht sich als kontroverses und agiles Modell, welches insbesondere auf sich **ändernde Anforderungen** bestmöglich reagieren kann.

KENT BECK wurde im März 1996 Projektleiter im legendären Projekt **C3** bei Daimler Chrysler. Zusammen mit Ward Cunningham und Ron Jeffries sollte - auf der Basis von Smalltalk und GemStone - ein einheitliches Bezahlungssystem entwickelt werden.

Interessanterweise ging bei diesem Projekt einiges schief. Das System erfüllte seine Erwartungen nicht. Das Projekt wurde am 1.2.2000 gestoppt. Während dieser Zeit erfuhr Kent Beck daher schmerzlich, dass in Softwareprojekten vieles schief läuft. Gleichzeitig aber sind viele Prinzipien guter Softwareentwicklung - wie z. B. das Testen - bekannt werden aber nicht ausreichend und gut genug genutzt. Aus dieser Motivation entwickelte und publizierte **KENT BECK** XP. Das legendäre Buch „*Extreme Programming Explained*“ wurde 1999 publiziert.

Auf der XP-Konferenz im Jahre 2000 kündigte ein Daimler Chrysler Manager sogar an, XP intern zu verbannen. Aber nur einige Zeit später führte Daimler Chrysler die XP-Methoden wieder ein.



Hinweis

Ziele von XP

- XP versucht den **Menschen** in den Vordergrund zu stellen und so seine Produktivität zu erhöhen
- XP versucht die **sozialen** Aspekte der Interaktion von Menschen im Projekt zu berücksichtigen
- XP versucht ständig die Möglichkeit zu öffnen, das Projekt, den Prozess und den Code zu verbessern
- XP versucht einen neuen Stil zu etablieren, wie mit Code umgegangen wird
- Die Art, wie Software entwickelt wird, wird durch viele später vorgestellte Prinzipien stark umgestellt

5.1 XP Prinzipien

Bevor es zu den spannenden und praktischen Prinzipien von XP geht, zunächst noch etwas über die Grundwerte von XP. Diese klingen zwar recht allgemein, es ist jedoch interessant zu beobachten, wie diese in den **nachfolgenden Prinzipien umgesetzt** werden.

Die nachstehenden Prinzipien zeigen auch, was in der Vergangenheit in „fat“ oder wasserfallartigen Vorgehensmodellen schief gegangen sein kann.

- **Communication:** Kommunikation ist - nicht nur in Projekten - sehr wichtig. Der „alten“ und herkömmlichen Denkweise zufolge müssen Definitionen und Abstimmungen eher auch über **Dokumente** vorgenommen werden. Es war also ein sehr dokumentenzentriertes Vorgehen. Unter XP sollen alle Personen enger zusammengeschaltet werden und - auch über den Code - eine Art **Community** mit „**shared knowledge**“ bilden. Dazu tragen mehrere Faktoren, wie ständige Kommunikation, die Verwendung von Metaphern, ständige Kooperation mit dem Anwender und gefördertes Feedback bei.



- **Simplicity** - (Einfachheit) ist ein Prinzip und zugleich auch die einzige Aktivität, die in den nächsten Seiten bei der Beschreibung der Aktivitäten auftaucht. Im Projekt fokussiert man sich zunächst auf die **einfachste Funktionalität**. Alles was implementiert wird, wird so einfach wie möglich gehalten. Eine Weiterentwicklung wird dann mittels Refactoring vorgenommen. Das bedeutet, dass man nicht besonders vorausschauend arbeitet und initial das Enddesign definiert, sondern zu dem Enddesign hin „evolutioniert“. Hier setzt auch einer der heftigsten Kritikpunkte an, dass das Refactoring zu einem Design teurer und aufwendiger sein kann, als die initiale Festlegung eines Designs.



- **Feedback** - nichts ist wichtiger.
 1. Feedback vom Kunden (**Customer**)
 2. Feedback vom Team, bspw. Informationen über die Dauer von Entwicklungen, internes Feedback, etc.
 3. Feedback durch den Produktionszyklus, indem die Tests ausgewertet werden. Änderungen werden dadurch mit überwacht.

- **Courage** - im Projekt heißt dies Mut zu zeigen. Den Mut, nicht sehr vorausschauend zu programmieren. Mut seinen Code so zu refaktorisieren, dass er dennoch zukünftig besser angepasst werden kann. Den Mut eigene Design-Sichtweise mit Kollegen abzugleichen und so eine objektivere Lösung zu finden.



- **Respect:** Dieser Punkt war zu Beginn nicht in Kent Becks Prinzipien. Dennoch ist er bald als fünfter Punkt hinzugekommen. Respekt kann hierbei heißen:

- **Niemals** Code beisteuern (contribute), der das System lahmlegt, offensichtliche **Fehler** enthält oder nicht kompiliert
- Keinen Code beisteuern, der die **Tests** nicht besteht
- Die andere **Entwickler** nicht lahmlegen oder ihnen **Zeit stehlen**
- Die **Sichtweise** anderer anerkennen und nach der **besten Lösung** zu suchen. Dann versuchen diese Lösungen mittels Refactoring umzusetzen.

Betrachten wir nun die 28 „spannenden“ Regeln des XP!

5.2 XP Rules - Kategorie: Planung

Die Reihenfolge lehnt sich der XP Erläuterung im Web an und nicht an die ursprüngliche Reihenfolge wie sie in der Literatur angegeben ist.

www.extremeprogramming.org

1. User Stories = Idee eines „Use Cases“

In einem der ersten Projektmeetings werden Ideen von Use Cases gesammelt und auf **Storycards** festgehalten. Ziel ist die einfache Erfassung eines Use Cases, um den Aufwand für dessen Realisierung abzuschätzen und die Release-Planung voranzutreiben. Er besteht meistens aus ca. 3 Sätzen auf einer Karteikarte. Also gerade keine langen Requirements-Dokumente! Sie enthalten keine technischen Details (keine Technologien, Algorithmen oder E/R-Modelle), können bei Bedarf einfach **zerrissen** und wieder neu geschrieben werden, falls sich bspw. die „Story“ als nicht korrekt erwiesen hat oder einfach nur angepasst werden muss.

Storycards

Keine technischen
Details

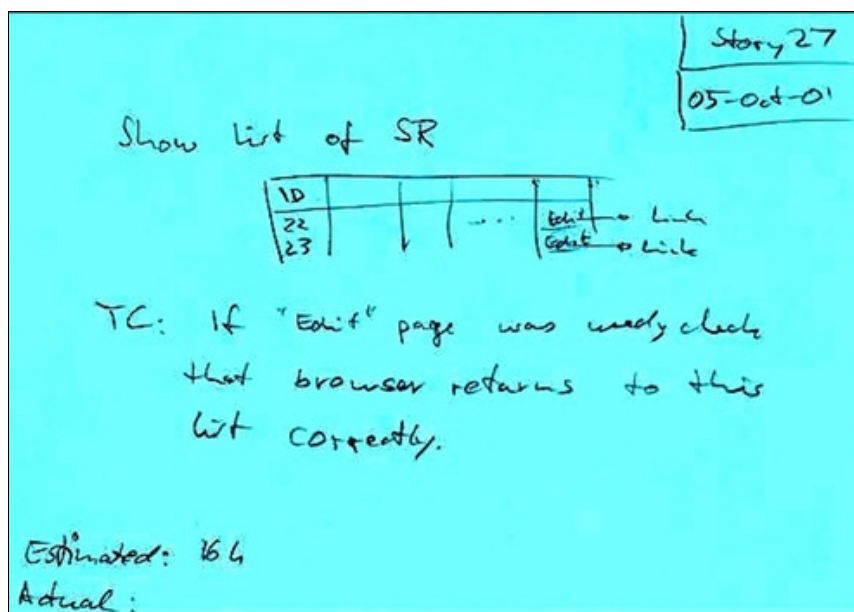


Abb.: Beispiel einer Storycard

Eine „Story“ kann auch Motor für einen Akzeptanztest sein. Die Storykarte soll gerade nicht so detailliert sein. Sie sollte jedoch für eine Aufwandsschätzung ausreichend sein. Wenn es später an die Implementierung geht, ist allerdings eine direkte Beschreibung vom Customer von größerem Wert.

Üblicherweise wird abgeschätzt, wie lange die Story für ihre Entwicklung idealerweise benötigt. Ist diese Zeit > 3 Wochen, dann sollte die Story wiederum aufgebrochen werden. Dies soll die folgende Abbildung veranschaulichen.

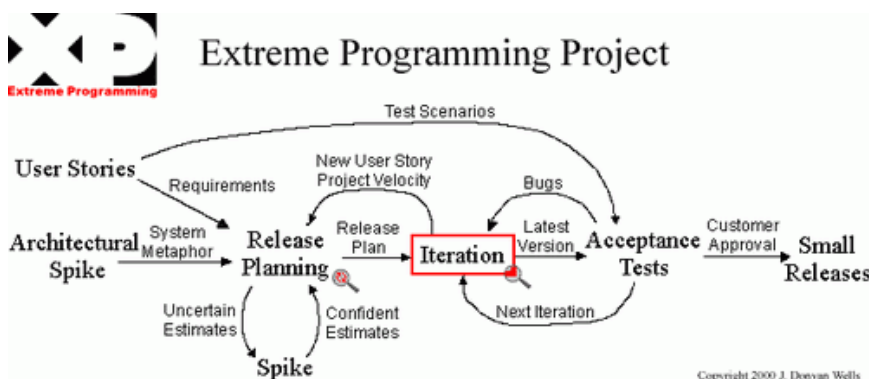



Abb.: Wie User Stories in die Planung eingehen

Das Symbol  wird auf den folgenden Seiten verwendet, um ein besonders bekanntes, zentrales oder auch kontroverses Feature von XP zu markieren.

5.3 XP Rules - Kategorie: Planung (Fortsetzung)

2. Release Planung

Enthält Iterationen konstanter Länge

XP sieht die Erstellung eines Release Plans vor. Ein **Release Plan** enthält die **Stories**, die für einen Projektstand zu einer bestimmten Zeit / **Milestone** erstellt werden sollen. Der Release Plan enthält wiederum **Iterationen**, d. h. Zeitelemente / -zyklen, in denen der Erfolg gemessen wird und Anpassungen vorgenommen werden können. Diese Iterationen sollten eine **konstante Länge** haben (z. B. 1 Woche). Iterationen sollten dabei nicht länger als eine Einheit vorausgeplant werden. Dies ermöglicht es, auf sich ändernde **Requirements** optimal zu reagieren.

Die Release Planung ist ein Tribut an die technische Planung und an die Business Planung, die sowohl **Ressourcen**, als auch die Finanzen planen und überwachen muss.

3. Small Releases = kleine Releasezyklen

Kleine Zyklen -> wendigeres Projekt

Ziel von kleinen Releases ist es, neue und kleine Funktionalitätseinheiten schnell zu realisieren und diese schnell dem Customer zu zeigen. Dies ermöglicht ein Feedback das so früh extrem wertvoll ist und in die weitere Planung / **Kodierung** mit einbezogen werden kann. Kundenwünsche erst später umsetzen zu können ist teurer und ggf. ist dafür einfach kein Geld oder keine Zeit mehr vorhanden.

4. Projekt Velocity = Maß für den Projektfortschritt

Zahl der User-Stories pro Iteration -> Leistung / Heartbeat des Projektfortschrittes

Zur Messung des Projektfortschrittes wird die Anzahl der **User-Stories** / technischen Tasks (Aufgaben) genommen, die in dieser Iteration fertiggestellt wurden. In den Meetings, in denen die Iterationen geplant werden, kann der Customer / Auftraggeber die gleiche Anzahl von Stories zur Entwicklung „in Auftrag geben“, wie in der vorangegangenen Iteration. Die Erfahrungen in den Iterationen sollten genutzt werden, um auch die übergeordneten Release Pläne anzupassen.

Ein wichtiger und schwieriger Punkt ist die **initiale Planung** der ersten Iterationen, da man nicht weiß, wie problematisch die Aufgaben sind oder wie effektiv das Team sein wird. Hier empfiehlt Kent Beck bei den ersten Iterationen lieber ins „Blaue zu schießen“ und sich dann schnell anzupassen. So können dann konkret die Erfahrungen genutzt werden, um den Gesamtaufwand besser abschätzen zu können, besser als detaillierte Spezifikationsdokumente zu schreiben.

5. Iterationen + 6. Iterationsplanung

Auf Iterationen innerhalb eines Release wurde bereits eingegangen.

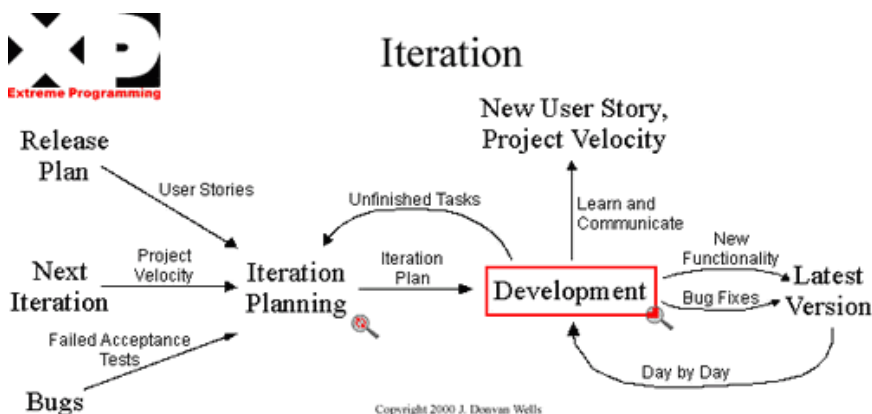


Abb.: Enthält Iterationen konstanter Länge

Was passiert in einer Iterationsplanung und in einer Iteration?

Die Iterationsplanung erzeugt einen Akzeptanztest. Scheitert dieser, so muss für die nächste Iteration angepasst werden. Aus den User-Stories werden Arbeitsschritte (Tasks) ermittelt, die auf Index- oder CRC-ähnlichen Karten festgehalten werden. Diese Tasks sind in der Sprache des Entwicklers (Developer) verfasst und stellen den Detailplan für die Iteration dar.

Die von den Entwicklern abgestimmten Arbeitsschritte enthalten konkrete Aufwandsabschätzungen. Es ist also wichtig, dass der Entwickler selbst den Aufwand festgelegt hat!

5.4 XP Rules - Kategorie: Planung (Fortsetzung)

7. Move people around


Cross Training
vermindert Projektrisiken

Personen im Projekt herumzubewegen klingt schwierig. Aber was ist gemeint?

- Entwickler übernehmen andere Aufgabenbereiche
- Entwickler üben die Hoheit über den Code von Anderen aus, dadurch sollen „Flaschenhälse“ vermieden werden. Entwickler können krank werden oder das Projekt verlassen.

KENT BECK bezeichnet dies als „**Cross Training**“, bei dem „**Wissensinseln**“, also die Konzentration von Wissen auf eine Person, vermieden werden. Ein wichtiges Prinzip das auch bei dem später beschriebenen **Pair Programming** angewandt wird.

8. Daily Stand Up Meeting

Fast jeder kennt den Ruf von traditionellen Meetings. Mittlerweise gibt schon  Spiele, bei denen man wie beim Bingo während eines Meetings „Buzzwords“ sammelt, und so gegeneinander spielen kann. Während der Besprechung!

KENT BECK stellt fest, dass in traditionellen Meetings viel Zeit für wenig Information verschwendet wird und schlägt folgende Alternative vor.



Jeden Morgen ein Meeting im Stehen durchführen!

Dies hat mehrere Vorteile. Normale Meetings werden nicht mehr so notwendig. Jeder versucht sich aus naheliegenden Gründen so **kurz** wie nötig zu **fassen**. Dies kann irgendwo geschehen oder sogar vor einem Rechner, um Dinge kurz zu veranschaulichen. Hier sollen wirklich kurz Probleme aufgezeigt und Lösungen vorgestellt werden.

Der Erfahrung des Modulautors nach funktionieren diese Meetings erstaunlich gut. Sie sind **fokussiert** und erzeugen trotzdem **Teamgeist** und Zusammengehörigkeitsgefühl. Diese Art von Meetings sind auf jeden Fall einen Versuch wert.

XP verbessern

9. Fix XP when it breaks

Die XP Regeln sind **kein Patentrezept**. Wenn diese nicht passen, sollten sie angepasst werden. Jedem Developer sollte jedoch klar sein, welche Regeln gelten und was man von Kollegen erwarten kann, bzw. selbst beizutragen hat. Stand-up-Meetings sind auch ideal dazu, Verbesserungen an XP im Projekt vorzuschlagen. Jedes Projekt und jedes Team sind individuell. XP kann nicht immer passen.

5.5 XP Rules - Kategorie: Coding

10. The customer is always available 🔥HOT

Ein Auftraggeber sollte ständig verfügbar / ansprechbar sein. Idealerweise sollte der Auftraggeber Teil des Teams sein, welches das Projekt durchführt und entwickelt. Dieser Punkt hängt besonders mit dem Prinzip der Kommunikation zusammen. In zu vielen IT-Projekten besteht noch heute ein unglaublicher Abstand zwischen Auftragnehmer und Auftraggeber. XP schlägt vor, einen Stakeholder wirklich zum **Teil des Projektes**, ja zum Mitarbeiter zu machen. Dies hat zwei ganz wichtige Vorteile:



- Erstens können Fragen vor Ort geklärt werden. Ein schneller und informeller Kontakt ist oft Gold wert. In der Regel wird dieser Auftraggeber im Projekt auch nicht alle Tage lang seine Meinung ändern, so dass man sich ärgert, nicht Requirements protokolliert zu haben.
- Zweitens, mit dem Auftraggeber im Boot wird es quasi zu einem internen Projekt des Auftraggebers. Immer wieder gibt es bei Projekten nach der Abgabe Ärger wegen vieler Nachforderungen und nicht selten enden die Parteien vor Gericht. Ist der Auftraggeber selbst Teil des Projektes, so wird er auch Teil der Lösung sein und im Streitfall zur Seite stehen. Die Mitverantwortung ist groß und ein Auftraggeber, der für das Projekt und somit auch für den Auftragnehmer agiert, ist wertvoll.

Die Nähe zum Auftraggeber macht es möglich, Dinge schnell zu klären.

11. Coding Standards

An Coding Standard, die zu Beginn abgestimmt wurden, sollten sich die Entwickler halten. Ein Beispiel sind die Java Code Conventions.

 <http://java.sun.com>

12. Code the Unit Test First 🔥HOT

Es sollte das Ziel sein, zuerst einen Test zu kodieren. Dieses Prinzip ist als „**Test-First**“ bekannt und wird in den Lerneinheiten zum Thema „Testen“ genauer betrachtet.

Es zeigt sich, dass **besser kodiert** wird (man quasi anders denkt / tickt), wenn Tests zuerst kodiert werden. Im extremen Fall nehmen diese Test eine Form von **Requirements** an, da sie sowohl das OO-Interface, als auch ein erwartetes Verhalten der Methoden einfordern.

Durch die Tests gibt es mit minimaler Anfangsinvestition gleich ein **Erfolgs Erlebnis**, wenn der Test erfolgreich abläuft.

Das **Design** wird durch den Test-First Ansatz **positiv** beeinflusst. Es wird so entwickelt, dass die Programme testbarer sind. Ein nachträgliches Einbeziehen von Tests ist ohnehin viel problematischer, als sie von vornherein durchzuführen.

5.6 XP Rules - Kategorie: Coding (Fortsetzung)

13. Pair Programming 🔥HOT

Dies ist wohl der bekannteste Punkt an XP. Leider hört man viel zu oft: „XP ist, wenn zwei am Rechner sitzen.“ Tatsächlich bedeutet diese Regel, dass zwei Entwickler am gleichen Rechner / Monitor entwickeln sollen.



KENT BECK bemerkt dazu, dass die Softwarequalität erhöht wird, ohne dabei die Zeit der Auslieferung zu beeinträchtigen wird. Es ist angeblich nicht intuitiv, aber 2 Personen liefern genauso viel Funktionalität mit höherer Qualität. Die höhere Qualität macht sich später im Projekt bezahlt.

Leider gibt Kent Beck keine quantifizierten Beweise für seine Behauptungen. Tatsächlich ist dies ein Streitpunkt aber auch ein interessanter Forschungspunkt geworden. Intensiv hat hier z. B. **WALTER TICHY** von der Universität Karlsruhe geforscht. [Mue05]

Empirische Erhebungen zeigen, dass es tatsächlich Aufgaben oder Projektkonstellationen gibt, bei denen die Annahme richtig ist. Jedoch nicht immer. Also kommt es bei der Bewertung dieser Frage auch hier wieder „darauf an“.

Natürlich gibt es noch andere kritische Aspekte. Es mag beispielsweise Chefs geben, die bei „pair programming“ annehmen, sie würden Leistung verlieren, weil einzeln programmieren ja „doppelt“ so effektiv sei.

Pair Programming ist auch nicht „extrem“ gemeint. Niemand kann jeden Arbeitstag im Jahr acht bis n Stunden täglich zusammen mit einem Kollegen am Rechner sitzen. Menschen brauchen auch Freiräume oder mal Privatsphäre. Wahrscheinlich kommt es auch hier in der Projektsituation auf den idealen „Mix“ aus pair programming und solo-programming oder solo-Zeit an.



Collective Code Ownership

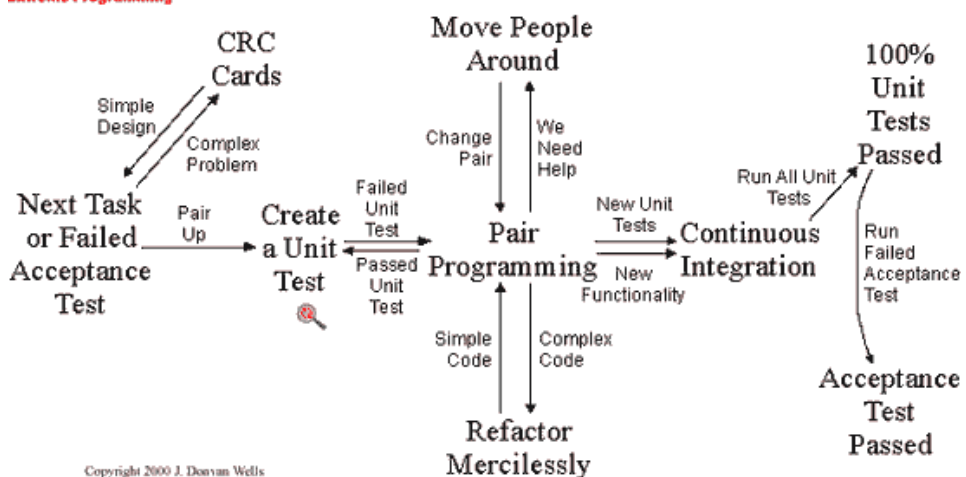


Abb.: Pair Programming im XP Kontext

Es lohnt sich sicherlich, Pair Programming einmal auszuprobieren. Mittlerweile gibt es sogar einige (virtuelle) Firmen, die komplett mit XP arbeiten und diesen Arbeitsstil leben.

5.7 XP Rules - Kategorie: Coding (Fortsetzung)

Nacheinander auf einem Release
Rechner
i Integrieren

14. Sequential Integration + 15. Integrate Often 🔥HOT

Neuer Code muss irgendwann integriert werden und das kann zum Problem werden. Wenn dies bspw. gleichzeitig in Versionskontroll-Repositories geschieht, kann die integrierte Software schon mal nicht mehr lauffähig sein.

Kent Beck hält hier eine dezidierte Person oder ein Team als Integrationsmanager als nicht angemessen. Stattdessen schlägt er eine sequentielle Integration vor, bei der auch die Idee der Code Community greift. Die Gruppe der Entwickler ist dafür zuständig, dass der integrierte Code lauffähig ist. Um sequentialisieren zu können, muss eine Art **Lock** eingeführt werden. Dies kann über ein einfaches Token geschehen, z. B. durch einen Release Computer, der als Token fungiert.

Es soll möglichst oft integriert werden. Am besten alle paar Stunden wenn möglich. Nur so können Entwicklungen verhindert werden, die auseinanderlaufen.

16. Collective Code Ownership 🔥HOT

Dieses Prinzip wurde bereits erwähnt. Der Code sollte allen gehören und Developer sollten sich nicht nur mit dem eigenen Code auseinandersetzen - sondern vielleicht auch Verantwortung für den Code der Anderen übernehmen. Das geht an einigen Stellen soweit, dass jedem erlaubt sein sollte, anderen Code zu fixen oder zu refactoren!

17. Optimize Last

Dieses Prinzip ist schon viel älter als XP. Dennoch wird es im Zuge mit XP genannt: Nicht initial optimieren. Es gibt schöne Beispiele, bei denen der Code beispielsweise initial auf Performance getrimmt wurde, dann aber das eigentliche Projektziel oder die Kernfunktionalität vor Optimierung nicht mehr erreicht wurde.

Daher sollte erst am Ende optimiert werden. Es sollte nie vorher geraten werden, was denn das „Bottleneck“ des Systems sein könnte. Erst am Ende zeigen sich oft die wahren Flaschenhälse.

18. No Overtime

Überstunden sollten nie eingefordert werden. Nach **KENT BECK** hilft dies wenig. Auch weiteres Personal dazu zu kaufen, ist nicht sinnvoll. Stattdessen sollte eine Release-Planungssitzung vorgenommen werden und lieber am Funktionsumfang gedreht werden.



5.8 XP Rules - Kategorie: Designing

19. Simplicity 🔥HOT

Design einfach halten

Beschreibt das Streben nach einem einfachen Design und der einfachsten Lösung, mit der die Aufgabe noch erfüllt werden kann. Komplexe Codes sollte man immer ersetzen.

Simplicity wurde als initiales Prinzip bereits vorgestellt.

20. Choose a System Methaphor 🔥HOT

Gute Metaphern für das System erleichtern den sprachlichen Umgang. Dies gilt sowohl für alle Granularitäten - vom Projektnamen über User Stories bis hin zu Klassen- und Methodennamen.

21. Use CRC Cards

CRC-Karten fördern das Design und helfen dabei weniger funktional zu denken. Die Karten selbst sind dann Objekte und können angefragt und auch wieder verwendet werden.

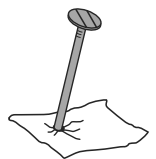
Part	
Responsibilities:	Collaborators:
Has a vendor cost, internal cost, quality and tax flag and belongs in inventory	Inventory Vendor
The part is associated with a work order when being purchases	WorkOrder PurchaseOrder
Parts go on charge slips	ChargeSlip
Parts are purchased from vendors	Vendor VendorPart

Abb.: CRC-Karte
Beispiel

22. Create a Spike Solution

(Technischer)
Durchstich

Hier bedeutet *Spike* so viel, wie das Problem festnageln. Die Idee ist eine bereits länger bekannte Weisheit und wird auf Deutsch meist „**(technischer) Durchstich**“ genannt. Wenn es irgendwelche technischen Probleme oder Fragestellungen in Bezug auf das Design gibt, soll ein schneller Prototyp gebaut werden, der von allem anderen abstrahiert, das Problem klärt und damit eine schnelle Antwort ermöglicht.



23. Never Add Functionality Early 🔥HOT

Ursprünglich gibt es eine weitere Anspielung auf Simplicity - den „**Business Value First**“. Es sollte mit der Kernfunktionalität begonnen werden und keine „goldenen Wasserhähne“ eingebaut werden. Also auch keine Funktionalität einfügen, von der man meint, sie würde später gebraucht werden. Hier wird oft die (80/20) Erfahrung angeführt, dass nur wenig von dem wirklich gebraucht wird, was nicht Kernfunktionalität ist.

24. Refactor Mercilessly 🔥HOT

Refactor Mercilessly bedeutet gnadenlose Refaktoren, d. h. Code umgestalten, umbauen, verbessern, etc. Alles was bereits gebaut wurde kann verändert werden. Code umzugestalten kostet Überwindung, aber es lohnt sich. Redundanzen rauszuwerfen, Designs zu verbessern und unnötige Codeteile zu entsorgen, lohnt immer. Dies ist jedoch oft nicht leicht, da es auch bedeutet, Fehler einzugestehen. Als Entwickler sollte man mutig genug sein, an Fehlern zu wachsen.

5.9 XP Rules - Kategorie: Testing

25. All code must have unit tests 🔥HOT

Neue Werkzeuge
für das Testen

Tests sind bereits im Prinzip „Test First“ angesprochen worden. Sie sind in XP viel mehr eine der Regeln / Prinzipien. Es beginnt damit das Kent Beck sich mit Kollegen zusammengetan hat, um das folgende Problem zu lösen.

Jeder weiß, dass Tests wichtig sind. Warum testet dann keiner? Ganz einfach, weil es noch nicht **Spaß** macht! Also wurde ein neues Testframework entwickelt, dass das Testen einerseits einfach macht und gleichzeitig „Fun“ ist. Ein Beispiel für ein solches Framework ist JUnit. JUnit wird in einer Lerneinheit hier in Softwaretechnik noch aufgeführt. Als Motivation dient bei dieser neuen Art zu testen ein grüner Balken, der angezeigt wie viele einer beliebig großen Menge von Tests positiv abgeschlossen werden konnten.

Der zweite wichtige Aspekt ist, dass die beliebig vielen Tests automatisiert ablaufen können. Sie können bei Bedarf sogar einen Report abliefern. Die Tests integrieren sich daher ideal in einen Zyklus, in dem schnell Prototypen erstellt werden und viel integriert wird.

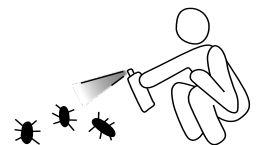
26. All code must pass unit tests before it can be released

Nur getesteter und korrekter Code kann released und integriert werden. Anderenfalls würde es die Kollegen lahmlegen und das Projekt verzögern.

27. When a bug is found tests are created

Fehler generell
und nicht punktuell
bekämpfen

Mindestens genauso wichtig wie die Beseitigung eines Fehlers (der nicht durch Tests entstanden ist) ist es, Tests zu entwickeln, die diesen und ähnliche Fehler zukünftig aufdecken können.



Dies gilt sowohl für Entwickler, die einen Programmfehler entdecken (und diesen dann soweit möglich mit weiteren JUnit Tests abdecken), als auch für andere Anwender, die einen Akzeptanztest erstellen, um eine perfekte Reproduktion des Tests zu ermöglichen.

28. Acceptance Tests

Anwender segnet
User Stories ab


Aus den User Stories werden Akzeptanztests erstellt (außerhalb von XP auch üblicherweise **funktionaler Test** genannt). Der Entwickler beschreibt einen oder mehrere Abläufe, die nach der korrekten Abarbeitung die „richtige“ Implementierung der User Story bestätigt. Diese Akzeptanztests können - wenn möglich - automatisiert werden oder - um effizienter zu sein - von den Entwicklern intern durchgeführt werden. Die endgültige Abnahme erfolgt aber nur vom Anwender.


5.10 XP kritisch betrachtet

Einige Prinzipien des XP provozieren und möchten das Althergebrachte einigermaßen umkrempeln. **KENT BECK** war klar, das XP kein Patentrezept ist. Was beispielsweise die Projektgröße angeht, so sagte **BECK** (hier übersetzt):

„XP ist nicht für Projekte mit 50 Personen geeignet und wahrscheinlich auch nicht für Projekte mit 20 Personen; aber bestimmt für 8 oder 15 Personen geeignet.“

Dennoch gibt es erfolgreiche XP-Projekte mit über 100 Entwicklern.

„Any one [XP] practice doesn't stand well on its own (with the possible exception of testing). They require the other practices to keep them in balance.“  [Be04]

Mittlerweile gibt es Bücher, die XP einer gründlichen Kritik unterziehen und Verbesserungen vorschlagen oder es in den rechten Kontext rücken.  [RS03]

Kritikpunkte und Dinge die zu beachten sind

- Unter Punkt 10 wurde angesprochen, dass man einen On-Site Customer vor Ort hat, welcher **Requests einfach ändern** und dies auch informell mitteilen kann. Dies ist natürlich kritisch. Auch formale Change Requests, die insbesondere von höherer Stelle abgesegnet sind, können hier unter Umständen viel vorteilhafter sein. Es ist daher darauf zu achten, dass **nicht** einfach so **unkontrollierte Changes** („scope creeps“) herausgeschossen werden, die später wieder unter großem Aufwand korrigiert werden müssen.
- XP ist das Gegenteil von dokumentenlastig. Es ist **dokumentenarm**. Dies geht beispielsweise nicht in einigen sicherheitskritischen Branchen wie Banken, da bereits gesetzliche Vorschriften oder Ämter (BASEL, BAKred) dies verhindern. User Stories, Tasks, CRC-Cards und Akzeptanztests sind die einzigen Dokumentationen. Dies ist strittig und kann in vielen Situationen fatal sein.
- Es gibt **kein vorausschauendes Design**. Einige Kritiker behaupten, ein guter Designer kann initial viel Refactoring und damit viel Zeit und Geld einsparen. Laut XP ist ein inkrementelles Design viel passender und besser, da sich sowieso alles ändern wird. Es hängt auch hier (wie so häufig) davon ab, wie sehr sich das zu erreichende Ziel bewegt oder ob z. B. kaum neue Erfahrungen im Projekt auftreten können.
- Das **Programmieren in Paaren** ist kontrovers. Einig sind sich jedoch fast alle, dass diese Art der Zusammenarbeit eine wertvolle Erfahrung ist und ausprobiert werden sollte.
- Der **On-Site-Customer** kann auch einfach mal „Mist bauen“, sodass der Auftrag im schlechtesten Falle scheitert. Ein alleiniger Ansprechpartner ist immer auch ein Single-Point-of-Failure, besonders wenn es sehr viele verschiedene Anwender gibt, deren Interessen dann natürlich nicht optimal berücksichtigt werden können.

6 Andere Agile Modelle

In diesem Kapitel werden andere agile Vorgehensmodelle vorgestellt.

Speziell werden die Modelle SCRUM www.controlchaos.com und Crystal Clear <http://alistair.cockburn.us> dargestellt.

Ein qualifiziertes Wissen über diese beiden Vorgehensmodelle ist nur mit intensiver Beschäftigung mit diesen Modellen und ausreichender Praxiserfahrung möglich.

Beide werden daher hier nur vorgestellt, um das Bild der agilen Modelle abzurunden. Dem Leser sollten diese Modelle von der Kategorie und vom Namen her bekannt sein. Ein tiefergehender Anspruch besteht nicht.

6.1 Crystal Clear



ALISTAIR COCKBURN

Crystal Clear <http://alistair.cockburn.us> wurde maßgeblich von ALISTAIR COCKBURN entwickelt und gehört ebenfalls zu den agilen Modellen.

Crystal Clear weist die folgenden Merkmale auf:

- Speziell für **kleine** Projekt**teams** ausgelegt
- Viele Mechanismen, um die **Kommunikation** im Team zu **verstärken**
- **Häufige Abgabetermine**
- Einfacher und unkomplizierter **Kontakt** mit dem **Endanwender**
- Es sind „**Reflection-Workshops**“ vorgesehen, die der kontinuierlichen Verbesserung des Entwicklungsprozesses dienen
- Nach jeder Iteration wird eine **Feedbackrunde** mit allen Projektbeteiligten durchgeführt. Auftretende Probleme im Prozess werden diskutiert und Lösungsmöglichkeiten erörtert und beschlossen.

COCKBURN sieht ein Softwareprojekt nicht als Engineering-Projekt, in dem initial Spezifikationen und Modelle erstellt werden müssen. Eher kommt das Projekt einem Spiel viel näher. **Spiele** sind viel **zielorientierter**, ebenfalls **kooperativ** und enthalten finite Zyklen (letzteres analog zu einem Adventure, in dem man sich im nächsten Level verbessern muss).

IT-Projekte
als Spiel

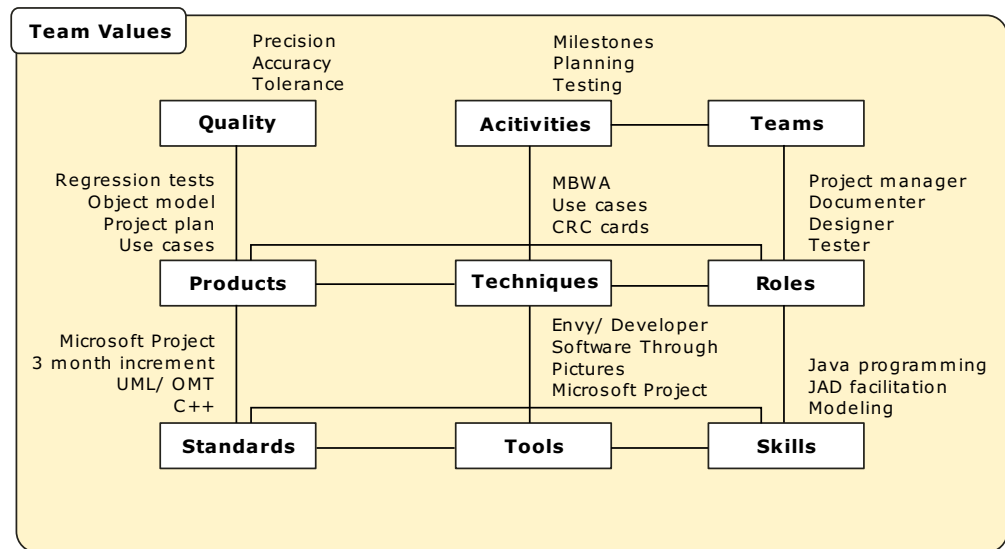


Abb.: Übersicht über einige
Elemente von
Crystal Clear

Crystal Orange
für größere Teams

Crystal Clear wurde für 3-8 Personen entwickelt. Es gibt jedoch mit **Crystal Orange** (COCKBURN teilt bestimmten Projektgrößen Farben zu) Anpassungen auf andere Projektgrößen.

6.2 SCRUM



KEN SCHWABER

Zu den bekanntesten agilen Methoden zählt Scrum, welche in den 1990er Jahren von Ken Schwaber, Jeff Sutherland und Mike Beedle entwickelt wurde. Es beinhaltet ein einfaches Prozessmodell, klar definierte Rollen und ein einfaches Regelwerk, mit deren Hilfe nicht nur große Projekte gesteuert, sondern sogar ganze Organisationen gemanagt werden können. Viele deutsche Unternehmen, darunter die Xing AG, die SAP AG und 1&1 Deutschland wenden Scrum mittlerweile erfolgreich an [[Ste10](#)] .

SCRUM wurde von **KEN SCHWABER** und **JEFF SUTHERLAND** entwickelt. Es hat den Ruf, ein „Hyper“-Produktivitätswerkzeug zu sein. Erstaunlicherweise wurde es anfänglich bereits 1986 (!) im Havard Business Review mit dem Titel „The new product development game“ dokumentiert.

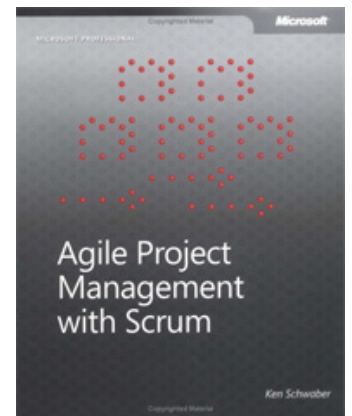
Sie nutzten das Wort Scrum aus dem Rugby-Sport, um zu beschreiben, dass es in der Produktentwicklung um die Zusammenarbeit mehrerer Fachdisziplinen geht. Auch steht dieses Bild für den Zusammenhalt, den ein Scrum-Team entwickeln kann und für nur wenige Regeln, die dafür notwendig sind. Ähnlich wie beim rauen Sport Rugby, sollen diese eingehalten und genau umgesetzt werden. [[Sch08](#)]

Der Begriff wurde dann durch verschiedene weitere Veröffentlichungen vor allem von Ken Schwaber, Jeff Sutherland und Mike Beedle als Framework zur Entwicklung von Software ausgebaut und später auch zur Organisation ganzer Unternehmen und Unternehmensbereiche weiterentwickelt.

Literaturreferenz:

[[Schw04](#)]

KEN SCHWABER: Agile Project Management with SCRUM



6.2.1 Scrum – ein Prozessmodell

Scrum ist ein Prozessmodell, welches sich auf den Prozessablauf konzentriert und keinen Einsatz spezieller Werkzeuge, wie zum Beispiel UML vorschreibt. Vielmehr steckt Scrum den Rahmen ab, in dem alle Aktivitäten der Produktentwicklung ablaufen. Dieser beinhaltet verschiedene Rollen, die nachfolgend, sowie spezielle Meetings und Artefakte, die anschließend in einer Kurzdarstellung des Scrum-Prozessablaufes näher erläutert werden.

Die Rollen in Scrum

1. Product Owner

Der Product Owner ist der Repräsentant des Kunden und verwaltet die Anforderungen an die Entwicklung. Er entscheidet am Ende, welche Funktionalitäten entwickelt werden sollen.

2. Team

Je nach Projektgröße gibt es ein oder mehrere Entwicklungsteams. Das Team besteht aus fünf bis neun Personen, organisiert sich selbst und realisiert die Entwicklungsprodukte.

3. Scrum Master

Der Scrum Master ist prinzipiell nicht höher gestellt als das Team, hat aber im Prozess die Aufgabe, die Werte und Regeln von Scrum zu wahren und Hindernisse zu beseitigen.

4. Externe Rollen

Sie sind nicht Teil des eigentlichen Scrum-Prozesses, wirken aber auf diesen ein, indem sie zum Beispiel die Ergebnisse in Review Meetings kommentieren.

- Benutzer
- Käufer und /oder Verkaufspersonen
- höheres Management

6.2.2 Der Scrum-Prozess – eine Kurzdarstellung

Am Anfang des Scrum-Prozesses steht die Idee eines neuen Produktes, die der sogenannte Produkt Owner zu einer Produkt-Vision entwickelt hat. Allein oder mit Hilfe von Teammitgliedern erarbeitet er die Produkt-Funktionalitäten, welche als Product Backlog Items bezeichnet werden. Alle Product Backlog Items werden im sogenannten Product Backlog festgehalten. Ähnlich einem Lastenheft stellt der Product Backlog so die Grundlage der geforderten Leistung dar.

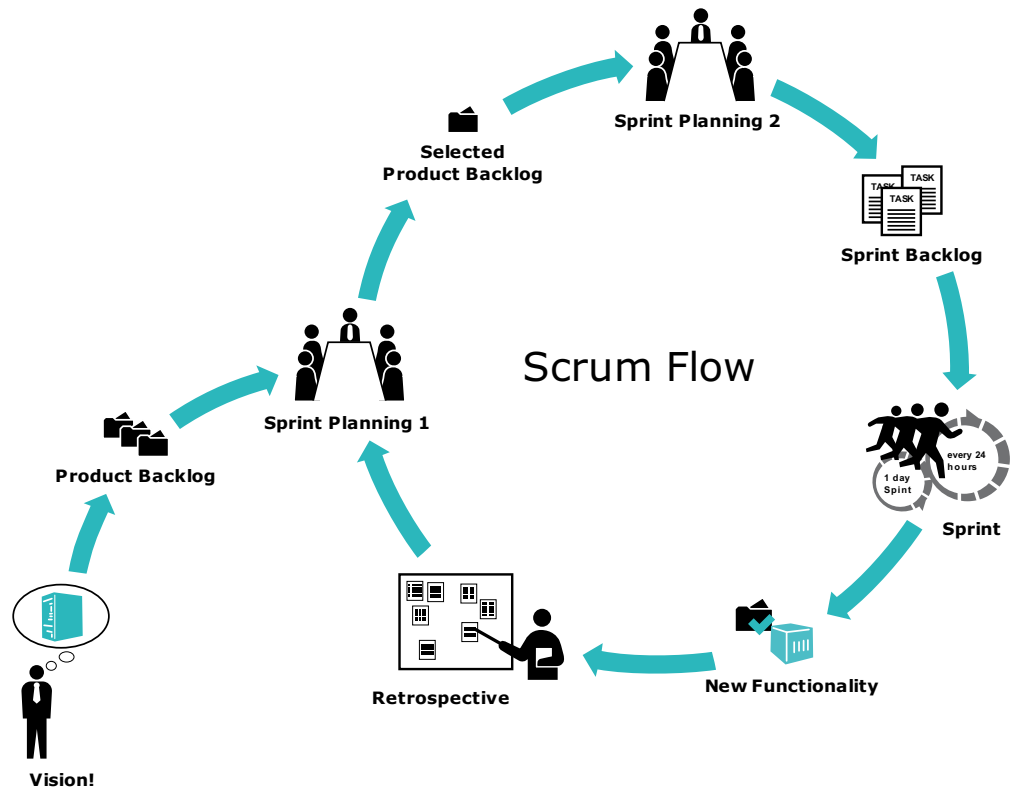


Abb.: Scrum – Prozessmodell

Für jedes Product Backlog Item wird der Aufwand von den Teammitgliedern geschätzt und durch die Reihenfolge die Priorität festgelegt. Es gibt verschiedene Arten der Aufwandsschätzung – eine davon ist beispielsweise das Planning Poker. Bei dieser Variante erfolgt die Aufwandsschätzung durch die Einteilung in Aufwands-Kategorien, zum Beispiel Kategorie 1 für »sehr geringer Aufwand« und Kategorie 13 für »extrem hoher Aufwand« sowie entsprechenden Kategorien dazwischen.

Nachdem der Product Backlog geschätzt ist, haben alle Teammitglieder eine Vorstellung davon, wie das gewünschte Produkt aussehen soll und der Product Owner eine erste Einschätzung vom finanziellen Aufwand.

Priorität	Nr.	Beschreibung	Aufwandsschätzung
Sehr hoch	12	Bugfix: Speicherung der Adressdaten	2
even	15	Single-Sign-On	13
Hoch	14	XML-Export der Kontaktdaten	1
even	7	Performancetuning	3
Mittel	5	Setup erstellen	1

Tab.: Beispiel für einen Product Backlog

Sobald das Projekt genehmigt ist, nimmt das Entwicklungsteam seine Arbeit auf. Dafür wird die gesamte Projektdauer in klar abgegrenzte zeitliche Intervalle, in sogenannte Sprints, eingeteilt. Jeder Sprint soll etwa zwei bis vier Wochen umfassen – in dieser Zeit muss das Team seinen Teil des Product Backlogs in auslieferungsfähige Software, die sogenannte usable Software entwickeln.

Am Anfang eines jeden Sprints steht die taktische Planung, welche in zwei Planungsmeetings aufgeteilt ist. Beim Sprint Planning Meeting 1 werden vom Product Owner, dem Team, dem Management und den Anwendern des Produktes die Ziele des Sprints, dem Sprint Goal festgelegt. Sie wählen die höchst-priorisierten Product Backlog Items aus, welche auch tatsächlich in dem Sprint geliefert werden können. Die Summe dieser vereinbarten Product Backlog Items ist das Selected Product Backlog.

Im darauf folgenden Sprint Planning Meeting 2 diskutiert das Team untereinander die Aufgaben, die sich aus dem Selected Product Backlog ergeben und entwickelt einen Umsetzungsplan, den sogenannten Sprint Backlog, der konkrete Lösungsansätze und Detailaufgaben in beinhaltet.

Bei der anschließenden Umsetzung des Sprint Backlogs stimmen die Teammitglieder ihre täglichen Aufgaben untereinander in einem sogenannten Daily Scrum ab. Das Daily Scrum Meeting ist ein informelles Meeting von maximal 15 Minuten.

In diesem Meeting soll bekannt gegeben werden:

- Was habe ich seit dem letzten Daily Scrum erreicht?
- Was will ich bis zum nächsten Daily Scrum erreichen?
- Welche Impediments (Hindernisse) sind mir dabei im Weg?

Jeden Tag wird außerdem das Burndown Chart aktualisiert: dafür schätzt jedes Teammitglied seinen noch zu erbringenden Restaufwand seiner Tasks. Dieses Diagramm wird so von Tag zu Tag konkreter und ermöglicht eine Fortschrittanalyse für den aktuellen Sprint.



Abb.: Burndown Chart

Noch während des Sprints bereiten die Teammitglieder, gemeinsam mit dem Product Owner, die nächsten Sprints vor. Das Product Backlog wird aktualisiert und es werden für neue Backlog Items Schätzungen erstellt und gegebenenfalls aktualisiert.

Im Sprint Review wird am Ende des Sprints der Fortschritt anhand der entwickelten usable Software demonstriert. Diese Demonstration führt in der Regel wiederum zu neuen Ideen und zeigt deutlich den Projektfortschritt. Ist die Functionality, also die Funktionalität des Produktes dann soweit fortgeschritten, dass das Produkt an die Produktion freigegeben werden kann, werden im Anschluss die entsprechenden Maßnahmen durchgeführt.


Nach dem Sprint Review führt das Team die Sprint Retrospective durch, in der die Vorgehensweisen der Teammitglieder und Probleme diskutiert werden, um diese in den folgenden Sprints effizienter und angenehmer zu gestalten. 📖 [Glo08]


6.2.3 Scrum umsetzen

Um eine bessere Vorstellung davon zu bekommen wie Scrum umgesetzt werden kann, werden im Folgenden die Anforderungen an das Scrum-Team erläutert und eine Auswahl an Tools vorgestellt.


Scrum Werte

Das erfolgreiche Anwenden von Scrum ist ein Lernprozess, der Selbstorganisation, Initiative und Kommunikation verlangt. Der programmierende Außenseiter (Nerd), der nur im Stillen Kämmerlein sein Aufgabenpaket abarbeitet, ist dabei nicht gefragt.

Denn bei einem Scrum-Team geht es nicht nur darum, am Ende funktionierenden Code zu liefern, sondern ein Stück Produkt in seiner Gesamtheit.  [Glo08] Dafür ist Multidisziplinarität und fachübergreifendes Arbeiten notwendig. Jedes Teammitglied soll seine Kenntnisse und seine Kreativität für ein gemeinsames Ziel einbringen.

5 Werte für die Zusammenarbeit
 [SB08]

KEN SCHWABER UND MIKE BEEDLE haben für die Zusammenarbeit fünf Werte festgelegt. Diese betreffen nicht nur das Entwickler-Team, sondern alle Beteiligten:

1. Verpflichtung
Die Teams sollen sagen, was sie liefern können und liefern, was sie zugesagt haben
 [Glo08] .
2. Fokus
Scrum fördert und fordert die Konzentration auf die wesentlichen Aufgaben.
3. Offenheit
Durch Scrum soll Transparenz erreicht und Wissensmanagement betrieben werden: jeder soll sein Wissen weitergeben, statt es für sich zu behalten.
4. Respekt
Die Kommunikation und Handlungsweise untereinander soll respektvoll sein.
5. Mut
Scrum unterstützt nicht nur, sondern fordert auch den Mut, über Probleme und Hindernisse zu und um Hilfe zu erbitten oder anzubieten.

Diese Werte sind der eigentliche Kern von Scrum – nur sie erst ermöglichen das Funktionieren von Scrum. Sie in einer Unternehmenskultur tatsächlich und vollständig umzusetzen, erfordert viel Zeit und Geduld und wird sicherlich nicht beim ersten Sprint auf Anhieb klappen.

Durch eine reflektierte Arbeitsweise und beispielsweise dem Erfahrungsaustausch am Ende eines Sprints in der Sprint Retrospective, können diese Werte Schritt für Schritt umgesetzt werden.

6.2.4 Scrum Tools

Damit alle Beteiligten einen Überblick über das Projekt haben, ist ein geeignetes Werkzeug unerlässlich. An erster Stelle der Visualisierung steht dabei das sogenannte Taskboard, auf dem die Backlog Items dargestellt werden. Daneben gibt es weitere Artefakte, wie zum Beispiel das Burndown-Chart oder der Impediment Backlog, welche regelmäßig aktualisiert werden.

Im Folgenden soll nun das Taskboard näher vorgestellt werden. Auf diesem wird der aktuelle Bearbeitungsstand aller Backlog Items festgehalten. Die Backlog Items können dabei in Form von User Stories formuliert werden. Das bedeutet, dass die Software-Anforderung bewusst knapp gehalten und in maximal zwei Sätzen formuliert wird. Zu Beginn werden die User Stories in konkrete Tätigkeiten, sogenannte Tasks aufgeteilt.

Im Laufe des Sprints werden diese Tasks je nach Bearbeitungsstand dann einer der drei Spalten zugeordnet: »Not Done« bzw. »Noch nicht begonnen«, »In Progress« bzw. »In Bearbeitung« und »Done« bzw. »Fertig«.

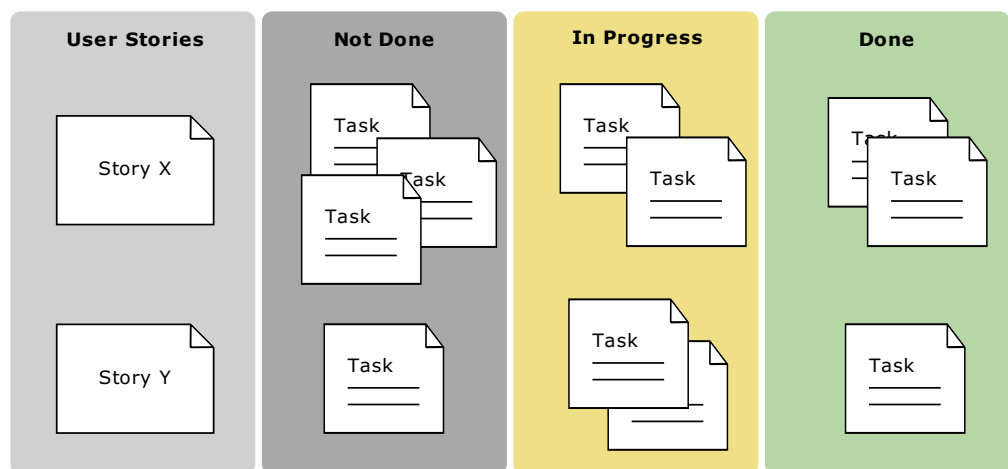


Abb.: Taskboard

Die einfachsten Mittel, um das Taskboard zu visualisieren sind die Pinnwand, das Whiteboard, die Magnettafel oder lediglich ein Bogen Packpapier. Mit diversen farbigen Pins, Stickern oder Magneten können zum Beispiel verschiedene Prioritäten signalisiert werden.

Diese Tafeln stehen direkt in Sichtweite des Scrum-Teams und werden jeden Tag nach dem Daily Scrum aktualisiert. Bei verteilten Teams können Webcams verwendet werden oder nach jedem Daily Scrum ein Foto verschickt werden.

Daneben verwenden einige Firmen Wikis oder verteilen das gute alte Spreadsheet. Zudem gibt mittlerweile eine sehr große Auswahl an Software-Tools darunter auch vielen Open-Source-Tools, welche zum Beispiel webbasiert sind und sich so auch für die Steuerung von (weltweit) verteilten Projekten eignen (► Siehe Links im Anhang).

6.2.5 Bewertung von Scrum

Unter den agilen Methoden nimmt Scrum zurzeit die führende Rolle ein – dies lässt sich begründen durch die zum Teil sogar messbaren Erfolgsfaktoren, von denen immer mehr Firmen profitieren möchten und deshalb Scrum einführen. Nichtsdestotrotz gibt es Kritiker, die die Freiheit von Scrum als Mangel an Vorgaben und diesen Ansatz daher für bestimmte Projekte als ungeeignet ansehen.

Erfolgsfaktoren

Wenn Scrum richtig umgesetzt wird, kann dies von großem Nutzen für ein erfolgreiches Projekt sein:


Probleme erkennen


- Probleme werden frühzeitig erkannt

Durch das Prinzip des Timeboxings, also der Softwareentwicklung innerhalb eines festgelegten Zeitraumes, wird mehr Effizienz bei der Implementierung erreicht: Der Priorität nach werden Anforderungen umgesetzt, bis das Ende der Timebox erreicht ist. An diesem Punkt werden Probleme bei der Entwicklung frühzeitig offensichtlich und es besteht die Möglichkeit, Lösungsoptionen zu entwickeln, entsprechende Maßnahmen zu ergreifen und das Projekt im weiteren Verlauf in die richtige Richtung zu lenken.

Zufriedenheit

- Verbesserung der Mitarbeiterzufriedenheit


Durch die aktive Beteiligung und Selbstorganisation steigt die Zufriedenheit der Mitarbeiter. Fest definierte Meetings wie zum Beispiel die Sprint Retrospective fördern die Zusammenarbeit im Team. Zudem führt die Freiheit, selbst einschätzen zu können, welche Arbeit innerhalb eines Sprints erledigt werden kann und die Arbeit ohne externe Kontrolle zu einer besseren Motivation  [Eri10] .

So beantworten 52 Prozent der Scrum-Projektmitarbeiter bei Yahoo! die Frage nach Verbesserung der Teammoral durch Scrum in einer Umfrage positiv  [DB07] .

Produktivität

- Erhöhung der Produktivität

Durch das Fokussieren auf die wichtigsten Anforderungen, den Erfahrungszuwachs von Sprint zu Sprint und das Vermeiden von Fehlentwicklungen durch aktive Beteiligung steigert Scrum die Produktivität. Zudem werden mit jedem Sprint fertige Produktteile geliefert, welche direkt getestet und integriert werden können.

Bei Yahoo! konnte beispielsweise eine Verbesserung der Produktivität von 68 Prozent durch den Einsatz von Scrum ermittelt werden  [DB07] .

Kundenzufriedenheit

- Erhöhung der Kundenzufriedenheit

Durch die enge Zusammenarbeit mit dem Kunden, in der Rolle des Product Owners und dessen Einbeziehung beispielsweise in Anforderungswshops und Sprint-Reviews wird sichergestellt, dass die resultierende Software die Kundenbedürfnisse befriedigt.

 [Pic08]

Und obwohl die Anforderungen zu Beginn eben nicht exakt festgeschrieben werden und somit die Verträge mit dem Kunden eher flexibel gestaltet werden müssen, zum Beispiel in Form von projektbezogenen Verträgen, haben Firmen wie Payback sogar gute Erfahrungen damit gemacht. Denn die Wahrscheinlichkeit, dass der bei einem Wasserfallprojekt vorher festgelegte Leistungsumfang tatsächlich ohne Nachbesserungen auskommt, ist ohnehin sehr gering.

 [Eri10]

Kritikpunkte

Mangel an Vorgaben

Trotz der genannten Erfolgsfaktoren gibt es Kritiker, die zum Teil genau die Punkte kritisieren, die von anderen hochgelobt werden. So wird die Freiheit von Scrum als Nachteil von denen angesehen, denen das Prozessmodell nicht ausreicht und die mehr Details für Planung und Design bzw. eine klare Entwicklungsmethodik fordern. Der Mangel an diesen Vorgaben führt nach Ansicht der Kritiker dazu, dass Scrum nicht für große bzw. verteilte und nicht für sicherheitskritische Projekte geeignet ist.

Mechanismen nicht ausreichend

Für größere Projekte würden mehr formalisierte Kommunikation und Koordinationsmechanismen über den verbalen Austausch durch Meetings hinaus benötigt. Zudem wären weitere – von Scrum nicht explizit vorgegebene – Planungsmechanismen für die Fehlerbehebung und Wartung von bestehenden Softwareanwendungen sowie der Kombination von Support und neuer Funktionen notwendig.

Sicherheit

Außerdem wären auch für sicherheitskritische Aspekte mehr Formalien notwendig, damit Architektur und Design nicht innerhalb jedes einzelnen Sprints vom Team festgelegt oder nachgebessert werden. Dies wäre zu risikoreich und ineffektiv und sollte stattdessen durch ein verbindliches Dokument sprintübergreifend fixiert werden 📖 [FD10] .

Klar ist sicherlich, dass Scrum allein kein Garant für erfolgreiche Projekte sein kann und dass der Erfolg stark von den individuellen Fähigkeiten und Motivationen aller Projektbeteiligten abhängt. Die selbstverantwortliche Arbeitsweise lässt sich nicht von jedem umsetzen. Manche mögen lieber Vorgaben, was er wie bis wann abzuarbeiten hat. Dies kann zu Konflikten führen und möglicherweise eine Mitarbeiterfluktuation zur Folge haben, wenn Scrum neu eingeführt wird.

6.2.6 Fazit

Im Gegensatz zu den klassischen Softwareentwicklungsmethoden beinhaltet Scrum einfache Prinzipien und ermöglicht viel Flexibilität im Entwicklungsprozess. Durch sein unkompliziertes Regelwerk ist es für nahezu jeden möglich diese Methode anzuwenden – praktische Werkzeuge unterstützen die Arbeit.

Bei richtiger Anwendung können die Werte der agilen Softwareentwicklung in hervorragender Weise gelebt werden. Dadurch kann sich der Einsatz von Scrum bei korrekter Anwendung nicht nur positiv auf die Zusammenarbeit mit dem Kunden auswirken und die Produktivität messbar verbessern, sondern auch noch die Mitarbeiterzufriedenheit besonders positiv beeinflussen.

So ist es nicht verwunderlich, dass Scrum mittlerweile die beliebteste Methode in der agilen Anwendungsentwicklung geworden ist. Der Marktforscher **FORRESTER** ermittelte im letzten Jahr, dass ein gutes Drittel von knapp 1300 befragten IT-Spezialisten heute mit agilen Methoden arbeitet – jede zehnte Firma setzt dabei Scrum ein. Als Grund wird die Einfachheit dieses Ansatzes genannt. Statt einer festgelegten Arbeitsweise in einem starren Ablauf stehen die Ergebnisse im Mittelpunkt. Die Wasserfallmethode findet dagegen nur noch in dreizehn Prozent der Firmen Anwendung.

Figur 1 Agile Adoption Has Reached Mainstream Proportions

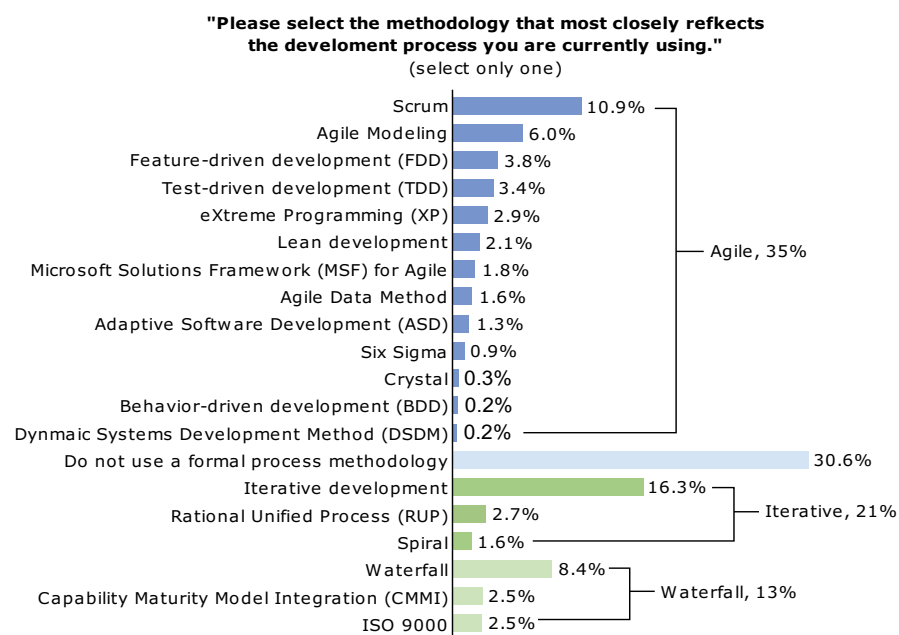



Abbildung: Forrester-Umfrage:
Welche Entwicklungsmethode
kommt Ihrer verwendeten am
nächsten?

[WG10]

Base: 1,298 IT professionals
Source: Forrester/Dr.Dobb's Global Developer Techographics Survey, Q3 2009

Bei einer Entscheidung für Scrum ist zu bedenken, dass dieses Prozessmodell lediglich ein Framework darstellt: es gibt Rahmenbedingungen in Form eines Ablaufes vor, der bestimmte Rollen, Artefakte und Meetings beinhaltet. Zudem bestimmt es Spielregeln für den zwischenmenschlichen Umgang. Was die Spieler, also die Projektbeteiligten daraus machen, hängt von ihren individuellen Kenntnissen, Fähigkeiten – und ihrem Willen ab. Die Disziplin, die notwendig ist, um die Scrum-Prinzipien bei der täglichen Arbeit einzuhalten, sollte nicht unterschätzt werden. Sie erfordert Zeit und Geduld. Dabei sind Offenheit, Kommunikation und Selbstverantwortung gefragt. Jeder hat nicht nur die Möglichkeit, sondern steht in der Pflicht, sich intensiv am Entwicklungsprozess zu beteiligen und als Teil zum großen Ganzen beizutragen.

Denn nur bei strikter Einhaltung der Regeln und Vermeiden von „Water-Scrum“, wie der Scrums Gründervater **JEFF SUTHERLAND** ein Zwischending aus agiler Anwendungsentwicklung und dem klassischen „Waterfall-Model“ bezeichnet, kann Scrum seine volle positive Wirkung erzielen  [Zei10] .

6.2.7 Ausgewählte Literatur zum Thema Scrum

Das Kapitel zu Scrum wurde von Frau Ina Fischer zusammengestellt.

Open Source Scrum Tools

Auflistung von **BORIS GLOGNER**:

 <http://borisglogner.com> (Abruf 15.11.2010)

Auflistung von **FLORIAN MARKERT**

 <http://www.scrum-tools.de> (Abruf 15.11.2010)

Quellen

DAVE WEST, TOM GRANT (2010): Agile Development: Mainstream Adoption Has Changed Agility.

Quelle:  <http://www.ca.com> (PDF 715 KB) (Abruf 15.11.2010)

ERIKSDOTTER, H. (2010): Die Scrum-Erfahrungen bei Payback.


Quelle:  <http://www.cio.de> (Abruf 25.11.2010)

GLOGNER, B. (2008): Scrum - Produkte zuverlässig und schnell entwickeln.

München: Carl Hanser Verlag.

HIROTAKA TAKEUCHI, IKUJIRO NONAKA (1986): New New Product Development Game. Harvard BusinessReview , S. 10.

ITEMIS AG (2010): Scrum kompakt.

Quelle:  <http://www.scrum-kompakt.de> (Abruf 15.11.2010)

KENT BECK, MIKE BEEDLE (2001): Agilemanifesto.

Quelle:  <http://agilemanifesto.org> (Abruf 15.11.2010)

KEN SCHWABER, MIKE BEEDLE (2008): Agile Software Development with Scrum. Deutschland: Pearson Studium


DEEMER, GABRIELLE BENEFIELD (2007): The Scrum Primer.

Quelle:  <http://www.rallydev.com> (PDF 439 KB) (Abruf 15.11.2010)

PICHLER, R. (2007): Scrum - Agiles Projektmanagement erfolgreich einsetzen. dpunkt.verlag.

SCHWABER, K. (2008): Scrum im Unternehmen. Deutschland: Microsoft Press.

STEVENS, P. (2010): Umfrage: Who is doing scrum in D-A-CH?

Quelle:  <http://www.doodle.com> (Abruf 15.11.2010)

WERNER FRITSCH, ANTON DECHKO (2010):

Scrum: Stärken und Schwächen eines Vorgehensmodells.

Quelle:  <http://www.crn.de> (Abruf 24.11.2010)

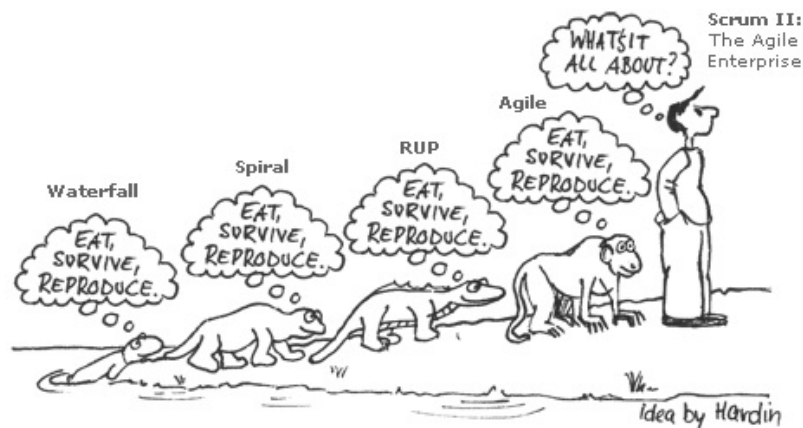
ZEITLER, N. (2010): Scrum braucht Disziplin.

 <http://www.cio.de> (Abruf 15.11.2010)

Zusammenfassung

- Das Vorgehensmodell ist die abstrakte und übergeordnete Antwort auf die Fragen, was, wie und womit durchzuführen ist.
- Das Prozessmodell - angewendet auf das Vorgehen bei der Erstellung von Software im Projekt - beschreibt die Schritte und Tätigkeiten die dabei anfallen.
- Der Prozessleitfaden ist ein Dokument, welches aus dem Vorgehensmodell abgeleitet und angepasst wird.
- Das Wasserfallmodell beschreibt den Ablauf des Entwicklungsprozesses in der Weise, dass die Ergebnisse einer Phase in die nächste Phase fallen.
- Spiralmodelle wurden zur Basis der inkrementellen Softwareentwicklung.
- Bei „Vorgehensmodell-Profis“ hat sich die Auffassung durchgesetzt, dass agilere Modelle den „alten“ Modellen in der Regel überlegen sind.
- Der RUP wird heutzutage eher als Vorgehensmodell für größere Projekte wahrgenommen und insbesondere in vielen Abwandlungen eingesetzt.
- RUP liegen sechs Erfahrungen zugrunde.
- Rational / IBM bietet mit dem RUPP ein Toolset an, welches alle Phasen und Aufgaben abdeckt und gut integriert.
- Wichtig beim RUP sind die vier Phasen: Inception, Elaboration, Construction und Transition.
- Eine unkritische und unangepasste Übernahme des RUP auf ein Projekt ist in der Regel teuer und ineffizient. Ein Tailoring ist nötig. Zudem hat sich gezeigt, dass eine Anwendung des RUP ohne das entsprechende Toolset ebenfalls kaum möglich ist.
- XP enthält praxisnahe Anleitungen für Entwickler und Projektleiter, die besonders Werte und die soziale Interaktion des Menschen in den Vordergrund stellen.
- Die XP-Prinzipien lauten: Communication, Simplicity, Feedback, Courage und Respect.
- Es gibt 28 XP-Rules. Davon sind etwa ein Dutzend wichtige Kernprinzipien.
- Crystal Clear und besonders Scrum sind wichtige Vertreter der agilen Modelle

Der Leser sollte nach den nachfolgenden Übungen zunächst „in der Materie“ sein und dann versuchen Praxiserfahrungen zu sammeln.



Wissensüberprüfung



Formulieren

Übung VOR-01

Begriffe

Beschreiben Sie kurz:

- Was bedeutet Vorgehensmodell? Womit beschäftigt es sich?
- Was ist ein Prozessmodell und ein Prozessleitfaden?
- Was haben Vorgehensmodelle gemeinsam?

Bearbeitungszeit: 15 Minuten



Formulieren

Übung VOR-02

Historie

Wo könnte ein Wasserfallmodell passen und wo nicht? Erfinden Sie zwei Projekte. Eines wo „der Wasserfall“ perfekt passt und eines, wo er überhaupt nicht passt.

Bearbeitungszeit: 20 Minuten



Formulieren

Übung VOR-03

Agile Modelle

Auf welche Schwierigkeiten wird man stoßen, wenn man in einem Unternehmen agile Modelle einführt, die dort bisher noch nicht gelebt wurden? Wie würden Sie diese Probleme beseitigen? Welche Vorteile würden Sie präsentieren?

Bearbeitungszeit: 15 Minuten



Formulieren

Übung VOR-04

RUP

1. Wie würden Sie die 6 RUP Erfahrungen in einem Projekt „leben“? D. h. erfinden Sie auch hier ein Projekt und erläutern Sie, wie Sie diese Erfahrungen konkret berücksichtigen würden!
2. Welche Phasen gibt es im RUP?

Bearbeitungszeit: 15 Minuten



Formulieren

Übung VOR-05

V-Modell

1. Wie wird die Qualität im V-Modell sichergestellt?
2. Was bedeutet „Tailoring“?
3. Mit welchen Methoden würden Sie im V-Modell Ihr Anwendungssystem beschreiben? Welche Methode halten Sie aus dieser Liste für die Beste? Warum?

Bearbeitungszeit: 20 Minuten



Formulieren

Übung VOR-06

Extreme Programming

1. Erfinden Sie ein Projekt, das ideal zu XP passt.
2. Stellen Sie sich vor, Sie sind Projektleiter in einem XP-Projekt, welches gerade anfängt. Was sind Ihre ersten Tätigkeiten?
3. Was ist der Vorteil von Story Cards? Und was der Nachteil?
4. Wie hängen Releases und Iterationen zusammen? Geben Sie ein Beispiel!
5. Was würden Sie als Projektleiter tun, wenn sich einige Entwickler gegen „move around“ und „pair programming“ sperren?
6. Ihr Auftraggeber wehrt den ersten Versuche ab, einen On-Site-Customer zu installieren. Ziehen Sie alle Register und überzeugen Sie ihn davon, dass dies sinnvoll ist!
7. Schreiben Sie eine CRC-Card für ein aktuelles, vergangenes oder zukünftiges Projekt.
8. Entwerfen Sie einen Akzeptanztest (1 Absatz) für die nächste größere Aufgabe, die Sie zu erledigen haben!

Bearbeitungszeit: 30 Minuten



Formulieren

Übung VOR-07

XP Applied

Versuchen Sie bitte, bei einem Projekt, welches Sie demnächst starten, möglichst viele XP-Prinzipien aktiv auszuprobieren. Sammeln Sie Ihre Erfahrungen und reflektieren Sie kritisch!

Bearbeitungszeit: 15 Minuten



Zuordnung

Übung VOR-08

Modelle

Ordnen Sie die Modelle den Zeitpunkten ihrer Entstehung zu.

60er	<input type="text"/>
70er	<input type="text"/>
1975	<input type="text"/>
80er	<input type="text"/>
90er	<input type="text"/>

Komponenten-/U-Modell	Wasserfallmodell
Phasen/Tätigkeiten	Phasenmodell
Spiralmodell	

? Test wiederholen Test auswerten

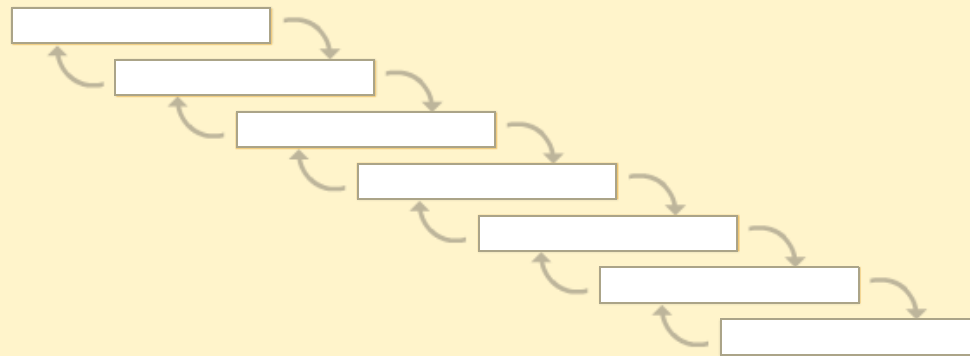


Zuordnung

Übung VOR-09

Wasserfallmodell

Ordnen Sie die Phasen des klassischen Wasserfallmodells zu.



Test	Betrieb
Codierung	Analyse
Softwareanforderung	Entwurf
Systemanforderung	

? Test wiederholen Test auswerten