

Prozessmodelle

Christof Kraschl, Cora Kumnig



Syp-Pre

HTL-VIllach 5AHIFS

05.02.2019

Inhaltsverzeichnis

[Klassische Prozessmodelle 2](#_Toc3297362)

[Spiralmodell 2](#_Toc3297363)

[Wasserfall 4](#_Toc3297364)

[Rup (Rational Unified Process) 5](#_Toc3297365)

[V-Modell 8](#_Toc3297366)

[Agile Prozessmodelle 10](#_Toc3297367)

[Scrum 10](#_Toc3297368)

[Extreme Programming 12](#_Toc3297369)

[Kanban 13](#_Toc3297370)

[Kanban Board 13](#_Toc3297371)

[Prinzipen und Praktiken 14](#_Toc3297372)

[Prinzipien und Praktiken 14](#_Toc3297373)

[Prototyping 15](#_Toc3297374)

**Prozessmodelle**

Agile und klassische Methoden bilden jeweils das Ende eines Spektrums im Rahmen der Softwareentwicklung. Klassische Methoden konzentrieren sich auf den formalisierten Prozess, während agile Methoden die eigentliche Entwicklung und die Kommunikation zwischen Entwickler und Kunde im Mittelpunkt sehen.

# Klassische Prozessmodelle

Klassische Methoden der Softwareentwicklung (heavyweight methods) übertragen ingenieurwissenschaftliche Ansätze der Projektdurchführung auf Softwareprojekte. Wie bei den „klassischen“ Ingenieurwissenschaften, wird hier viel Wert auf den Prozess gelegt. Dies bedeutet, dass die Planungsphase ausführlich ausfällt und allen Phasen eine detaillierte Dokumentation folgt, die wiederum als Eingabe für die darauffolgende Phase dient. Diesen Ansatz bezeichnet man auch als Software Engineering.

## Spiralmodell

*Das Spiralmodell ist ein iterativer Prozess, der die Kontrolle und Minimierung der Projektrisiken zum Ziel hat.*

Es ist es ein risikogetriebenes Vorgehensmodell für die Entwicklung. Es wurde 1986 von Barry W. Boehm beschrieben. Wichtig ist, dass es den Wartungsprozess nicht explizit betrachtet. Das Prinzip des Spiralmodells ist das wiederholte Durchlaufen (in Iterationen) von vier Schritten:

1. Risikoanalyse; Beschreibung von Rahmenbedingungen, Zielen und (Lösungs-)Alternativen
2. Evaluierung der (Lösungs-)Alternativen und auf dieser Basis Erkennen von Risiken und Ausarbeiten adäquater Strategien zur Minderung/Vermeidung etc.
3. In Abhängigkeit von den identifizierten Risiken, wird das Vorgehen für die nächste Iteration festgelegt.
4. Review der vorangegangenen Schritte und Planung der nächsten Iteration

Diese Schritte werden durch Quadranten visualisiert. Durch die iterative Vorgehensweise wird das zu entwickelnde System schrittweise ausgebaut. Eine Linie, die das Fortschreiten der Entwicklung kennzeichnet und in die Quadranten eingetragen wird, liefert das typische Spiralmuster, dem dieses Modell seinen Namen verdankt.

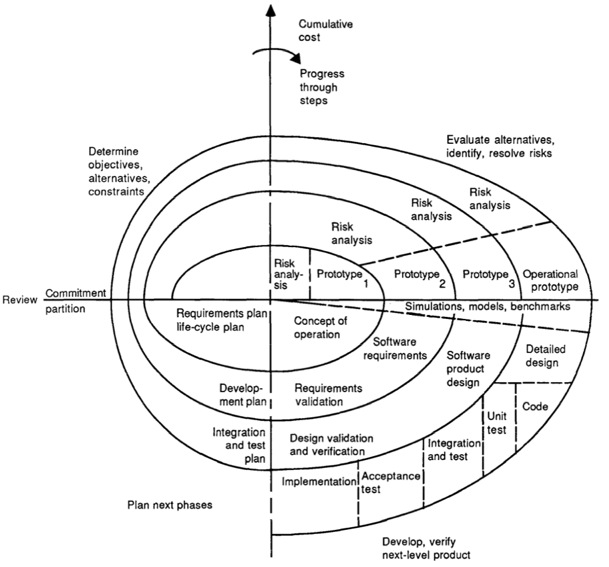


Abbildung 1: Spiralmodell

Zentral beim Spiralmodell ist die Erstellung von Prototypen, die eine kontinuierliche Prüfbarkeit des Systems erlauben. Die Prüfung, bzw. die Evaluierung, dient dazu, Risiken frühzeitig zu erkennen und angemessen darauf reagieren zu können. Die stetige Weiterentwicklung der Prototypen profitiert vom Wissenszuwachs, sodass davon ausgegangen wird, dass der Prototyp bis zum finalen, betriebsbereiten System wächst. Auch die anderen Produkte des Projekts, wie z.B. Architekturen, werden in jeder Iteration verfeinert.

Das Vorgehen, insbesondere für die Entwicklung der Software, kann in den einzelnen Iterationen variieren. Die Variation ist abhängig von den erkannten bzw. verbleibenden Risiken und muss im 3. Schritt für das weitere Vorgehen ausgewählt werden. Somit ist das Spiralmodell kein Vorgehensmodell im eigentlichen Sinne. Vielmehr ist es ein Rahmen, der in Abhängigkeit der Kritikalität mit konkreten Vorgehensweisen (Methoden) auszugestalten ist. Dabei kann der Fokus auf der Prototyprealisierung verbleiben oder aber auch ein Wasserfall-artiges Vorgehen zum Einsatz kommen.

Im 4. Schritt wird darüber hinaus auch ermittelt, ob ein Projekt in mehrere Teilprojekte in dem Sinne aufgeteilt werden kann, dass für einzelne Teilsysteme jeweils separate Spiralen etabliert werden. Insbesondere bei der Entwicklung von Produktfamilien kann das von Vorteil sein. Jedoch macht das Spiralmodell selbst keine Aussagen darüber, wie die Einzelteile/-projekte wieder zu synchronisieren sind.

## Wasserfall

Wasserfallmodel ist ein Vorgehensmodell, bei dem der Projektablauf in sequentielle Phasen gegliedert ist, deren Teilergebnisse aufeinander aufbauend zum vorher vollständig spezifizierten Projektergebnis führen. Das in diesem Sinne definierte Wasserfallmodell gilt als charakteristisch für das Traditionelle Projektmanagement bzw. wird mit diesem oft sogar gleichgesetzt.

Die grundlegende Idee des Wasserfallmodells ist die Aufteilung des Projekts anhand der Entwicklungstätigkeiten in Phasen. Folgende Phasen werden im Wasserfallmodell verwendet:

1. Analyse
2. Architekturentwurf
3. Implementierung
4. Test
5. Betrieb und Wartung

Das Wasserfallmodell gibt es in verschiedenen Ausprägungen. Die beiden bekanntesten sind das Phasenmodell nach Royce aus dem Jahre 1970 (einfache, ursprüngliche Version) und das Phasenmodell nach Boehm von 1981 (explizite Positionierung von Qualitätssicherung, feinere Gliederung der Anforderungserhebung und der Übergabe in den Betrieb). Die beiden Ausprägungen zeigen die gemeinsame Grundstruktur, aber auch die Abweichungen im Detail. Die Einordnung in die verschiedenen Phasen kann trotzdem problemlos erfolgen, obwohl die Ausgestaltungstiefe variiert.

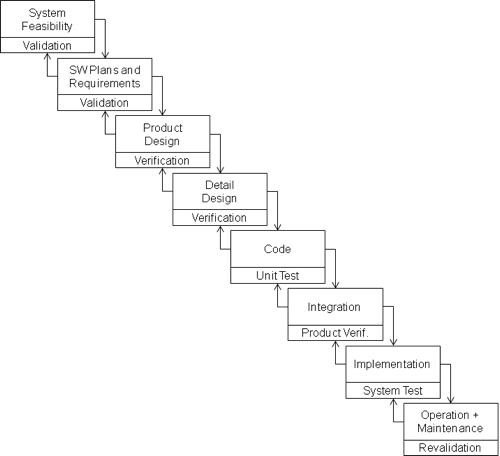


Abbildung 2: Wasserfallmodell nach Boehm

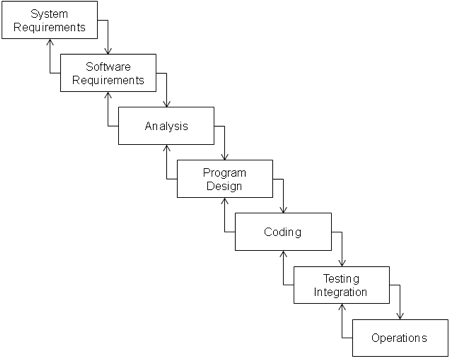


Abbildung 3: Wasserfallmodell nach Royce

Jede dieser Phasen schließt mit einem Meilenstein, zu dem eine definierte Ergebnismenge vorliegen muss. Jede Phase bündelt verschiedene Aktivitäten, die vollständig und in der richtigen Reihenfolge ausgeführt worden sein müssen. Am Ende jeder Aktivität liegt ein fertig gestelltes Ergebnis vor – in der Regel sind dies Dokumente, z.B. eine Architekturspezifikation. Da die Ergebnisse jeweils Eingaben für die folgenden Aktivitäten sind, müssen Aktivitäten auch vollständig abgeschlossen werden, bevor eine Folgeaktivität beginnt (eine zeitlich parallele Ausführung von Aktivitäten ist somit nur bedingt möglich).

## Rup (Rational Unified Process)

**Grundkonzepte und Phasen**

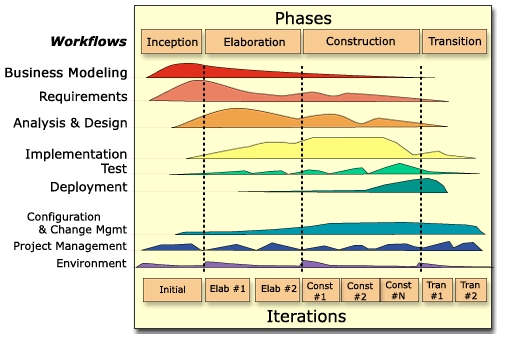
RUP deckt weite Teile des Softwarelebenszyklus ab. Er unterscheidet zwischen einzelnen Lebenszyklusabschnitten (Phasen) inkl. der Iterationen in den einzelnen Abschnitten. Orthogonal dazu sind die Disziplinen positioniert, die in einem Projekt nach RUP die Aktivitäten bündeln. Zusammen ergeben sie das bekannte RUP-Gebirge. Im Kern folgt der RUP den folgenden drei Grundprinzipien:

* RUP ist Anwendungsfall-getrieben.
* Die Architektur steht im Zentrum der Planung.
* Das Vorgehen zur Entwicklungszeit ist iterativ/inkrementell.

**Phasen und Meilensteine**

Abschnitte im Lebenszyklus eines Projekts bezeichnet RUP als Phasen. RUP untergliedert ein Projekt in die folgenden vier Phasen:

* Inception (Konzeptionsphase)
* Elaboration (Entwurfsphase)
* Construction (Konstruktionsphase)
* Transition (Übergabephase)



In jeder der vier Phasen werden Aktivitäten (Arbeitsschritte) einzelner Disziplinen gebündelt. Weiterhin sind die einzelnen Phasen in Iterationen unterteilt. Jede Iteration schließt mit einem Meilenstein (Business Decision Point) ab:

* Lifecycle Objectives
* Lifecycle Architecture
* Initial Operational Capability (Entwurfsmodelle und Beta-Release der Software)
* Product Release

Im ersten Meilenstein (lifecycle objectives) erwartet RUP die folgenden Ergebnisse: eine Projektvision inklusive eines rudimentären Anwendungsfallmodells, das die wesentliche Funktionalität des Zielsystems beschreibt. Zum Einsatz kommen hier üblicherweise UML-Anwendungsfalldiagramme. Weiterhin ist bereits ein erster, grober Architekturentwurf (z.B. mithilfe von UML-Komponenten- oder Klassendiagrammen) zu liefern. Organisatorisch soll hier bereits eine erste Identifikation relevanter Projektrisiken vorliegen.

Im zweiten Meilenstein (lifecycle architecture) ist ein Architekturprototyp vorzulegen. Weiterhin soll das initiale Anwendungsfallmodell weiter verfeinert werden. Organisatorisch ist zu diesem Meilenstein eine detaillierte Arbeitsplanung für die folgende Konstruktionsphase vorzulegen. Aus der Architektur sind daher bereits Arbeitspakete abzuleiten.

Der dritte Meilenstein (initial operational capability) fordert detaillierte Entwurfsmodelle (in verschiedenen UML-Diagrammtypen) und eine Beta-Version der Software. Während im vierten Meilenstein (product release) bereits eine produktive Software vorliegen soll.

**Disziplinen**

RUP definiert die folgenden Disziplinen:

* Business Modeling (Geschäftsprozessmodellierung) - Ziel ist das Erreichen des Verständnisses für die Geschäftsprozesse zu erreichen.
* Requirements (Anforderungsanalyse) - Die Anforderungen werden erfasst, dokumentiert, organisiert und im Projekt nachverfolgt.
* Analysis & Design - Die fachliche Architektur wird in ein technisches Design überführt.
* Implementation (Implementierung) - Es wird iterativ ein lauffähiges Softwaresystem erstellt
* Test - Das Softwaresystem wird getestet.
* Deployment (Auslieferung) - Das Softwaresystem wird an den Kunden ausgeliefert.



Diesen Kerndisziplinen ordnet RUP auch die Phasen zu, in denen sie schwerpunktmäßig bearbeitet werden. Unterstützende Arbeitsschritte, die unabhängig von einzelnen Phasen und somit querschnittlich im Projekt sind, ordnet RUP wie folgt ein:

* Configuration & Change Management (Konfigurations- und Änderungsmanagement) - Alle Projektartefakte, Änderungsforderungen und so weiter werden im System organisiert und verwaltet.
* Project Management - Das Projektmanagement wird im Projekt installiert und durchgesetzt.
* Environment (Infrastruktur) - Die Infrastruktur für das Projekt wird bereitgestellt und gepflegt.

Die Disziplin Environment hat den Schwerpunkt, sicherzustellen, dass im Projekt alle benötigte Infrastruktur vorhanden ist. Unter Infrastruktur wird im RUP eine abgestimmte Kombination aus Werkzeugen, Prozessen und Methoden verstanden, die das Team benötigt, um das Projekt erfolgreich bearbeiten zu können.

**Aktivitäten, Artefakte und Rollen**

RUP ist ein sog. aktivitätsgetriebener Entwicklungsprozess (im Gegensatz zum V-Modell XT, das ein produktzentrierter Entwicklungsprozess ist). Rollen, die im RUP definiert sind, sind also dafür zuständig, Aktivitäten durchzuführen, die im RUP detailliert beschrieben sind. Aktivitäten sind in festgelegten Reihenfolgen und Schritten zu absolvieren. Sie sind eindeutig bestimmten Phasen im Prozess zugeordnet. Aktivitäten erwarten bestimmte Artefakte als Eingabe und produzieren eine Menge festgelegter Artefakte als Ausgabe. Diese sind dann wieder Eingabe für Folgeaktivitäten. Die Kombination und Verknüpfung der Aktivitäten wird durch Workflows beschrieben, die gleichzeitig auch die Rollen, die mitwirken, einbinden.

## V-Modell

Das V-Modell ist vom Wasserfallmodell abgeleitet mit der Erweiterung, dass frühe Phasen mit späteren Phasen über Testfälle verbunden sind. Wichtige Bestandteile des V-Modells sind die Verifikation sowie die Validation der Teilprodukte.

Verifikation bedeutet die Überprüfung der Übereinstimmung zwischen einem Softwareprodukt und seiner Spezifikation, während unter Validation die Eignung eines Produktes bezogen auf seinen Einsatzzweck verstanden wird.

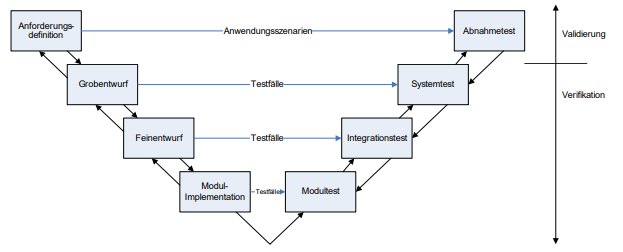
Verifikation: „Wird ein korrektes Produkt entwickelt?“

Validation: „Wird das richtige Produkt entwickelt?“

Generell werden im V-Modell für folgende Bereiche des Softwareentwicklungsprozesses Vorgehensweisen festgelegt:

* **Systemerstellung**: Hier werden Richtlinien für die Entwicklung und Dokumentation, sowie die Testfälle festgelegt
* **Qualitätssicherung**: In diesem Bereich geht es um die Festlegung von Qualitätsanforderung sowie Methoden zu deren Sicherstellung
* **Konfigurationsmanagement**: Im Konfigurationsmanagement geht es um die Verwaltung von Produkten, Rechten und Änderungen
* **Projektmanagement** Hier werden Vorgehensweisen zur Planung, Kontrolle, Steuerung und Koordination festgelegt.

Folgende Grafik zeigt das V-Modell aus dem Bereich der Systemerstellung:



Von der Anforderungsdefinition werden Anwendungsszenarien direkt an den Abnahmetest geleitet. Vom Grobentwurf werden Testfälle direkt an den Systemtest gegeben, ebenso wie vom Feinentwurf zum Integrationstest und von der Modulimplementation zum Modultest. Kommunikationsschritte können sowohl von der linken oberen Seite des "V" als auch von der rechten oberen Seite gestartet werden.

Das V-Modell deckt nicht nur den reinen Softwareentwicklungsprozess ab, sondern auch verwandte vor- und nachgereihte Prozesse wie z. B. Projektmanagement, Qualitätssicherung und Konfigurationsmanagement. Die einzelnen Bereiche des VModells beinhalten detaillierte Vorgaben, Ergebnismuster sowie Rollendefinitionen und –zuordnungen. Dadurch wird eine hoch standardisierte Abwicklung von Projekten ermöglicht. Das V-Modell ist im Gegensatz zum Wasserfallmodell auch für Großprojekte geeignet.

Der hohe Detailgrad bei den Standardisierungen des V-Modells wirft natürlich die Frage auf, ob es auch für kleinere Projekte geeignet ist, und ob es nicht zu einem Bürokratie-Overhead kommt. Zugleich besteht ein erhöhter Managementaufwand für V-Modell Projekte nicht zuletzt durch einen erhöhten Schulungsaufwand für die Projektmitarbeiter. Weiters ist die Verwendung von CASE-Tools beinahe obligatorisch, um die Vielzahl an Ergebnistypen und Prozessschritte zu modellieren.

Als weiteres Problem des V-Modells gilt die Tatsache, dass die gängigsten Versionen des Modells bestimmte Entwicklungsmethoden und –techniken fix vorgeben (Daten- und Funktionstrennung), wodurch andere Analyse- und Designmethoden unter Umständen außer Acht gelassen werden.

# Agile Prozessmodelle

Agile Methoden der Softwareentwicklung (lightweight methods) entstanden Mitte der 90er Jahre als Reaktion auf die bürokratischen, dokumentlastigen klassischen Methoden und deren Schwierigkeiten im Umgang mit sich schnell verändernden Anforderungen. Ihr Fokus liegt auf der eigentlichen Softwareentwicklung und nicht auf Prozess, Design oder Dokumentation. Um das Ziel der schnellen Anpassung an Änderungen zu erreichen, pflegen agile Methoden sehr kurze Iterationen, bei denen inkrementell entwickelt wird. Durch kurze Iterationen und kurze Release-Zyklen wird schnelles Feedback beim Kunden über den Fortschritt und Funktionsumfang einge-holt. Dieses schnelle Feedback dient zur Steuerung der weiteren Entwicklung. Im „Agile Manifesto“ drücken die führenden agilen Verfechter ihre Wertevorstellung der agilen Entwicklung aus. Das Agile Manifesto stellt Individuen und deren Interaktion über Prozesse und Pläne, d.h. die Konzentration der Methodologie findet auf individueller Ebene und nicht auf abstrakter Prozessebene statt.

Agiles Manifest:

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

## Scrum

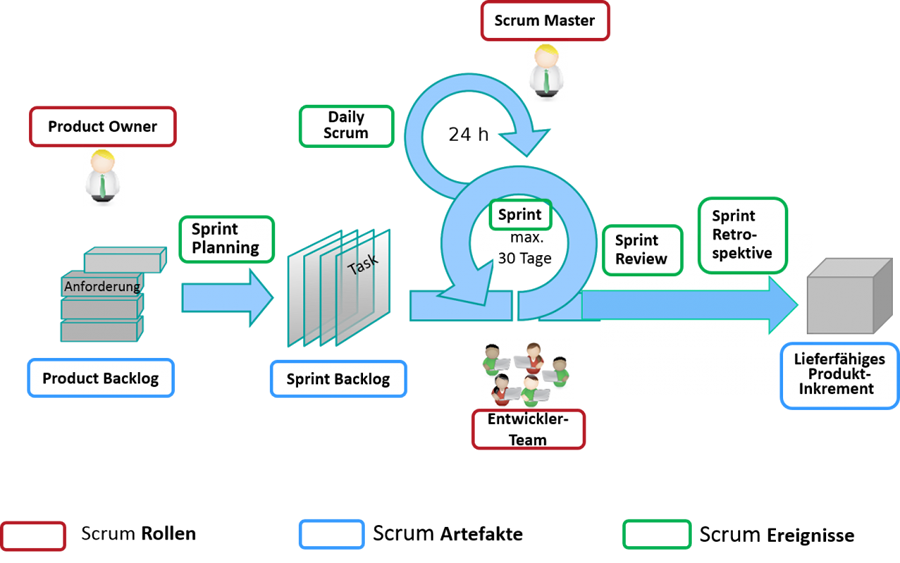
In Scrum organisieren sich Teams weitgehend selbstständig anhand gewisser Rituale (Daily Scrum etc.). Die Grundannahme von Scrum ist es, dass Projekte komplex und somit nicht von Anfang an detailliert planbar sind. Daher wird für das Projekt ein grober Rahmen vereinbart, in dem sich das Team selbstorganisierend bewegen kann.

Die vier grundlegenden Annahmen, auf denen Scrum aufbaut, entsprechen den Prinzipien der agilen Entwicklung. Scrum besitzt kein detailliert ausgeprägtes Rollenmodell, wie z.B. der Rational Unified Process oder das V-Modell XT, sondern verteilt die Aufgaben im Projekt auf drei Rollen. Die Verantwortung für das Ergebnis übernimmt ein sogenannter Product Owner, der im Wesentlichen mit dem Projektmanager zu vergleichen ist. Er wählt die umzusetzenden Anforderungen aus einem Product Backlog aus, priorisiert diese und sorgt für die Erreichung der gesetzten Ziele. Die Implementierung wird durch das Team durchgeführt, das selbstorganisierend innerhalb Sprints bestimmt, welche Elemente des Backlogs umgesetzt werden. Das Team hat dabei das Recht, eine Auswahl zu treffen, verpflichtet (commitment) sich dafür aber auch, dass durch die Auswahl gesetzte Ziel, zu erreichen. Der Scrum Master ist abschließend dafür zuständig, dass der Scrum-Prozess im Projekt umgesetzt wird. Er muss dafür sorgen, dass das Team produktiv arbeiten kann. Der Scrum Master ist kein Mitglied des Projektteams. Er hat keine Weisungsbefugnisse bezüglich des Projektgegenstands und mischt sich nicht in die Kommunikation zwischen Projektteam und Product Owner ein. Allerdings hat er die Pflicht, darauf zu achten, dass das Projektteam sich ohne Einflussnahme des Product Owners selbst organisieren kann. Somit ist der Scrum Master eine Art Mentor und Kontrollinstanz für das Projekt in Prozessfragen.

Genau wie beim Rollenmodell ist Scrum auch in Bezug auf die Artefakte und das Prozessmodell sehr schmal. Im Wesentlichen fokussiert Scrum das gesamte Projekt auf das Product Backlog und das Sprint Backlog. Ersteres enthält alle Anforderungen an die zu realisierende Software (Features). Das Product Backlog muss nicht vollständig sein und kann mit dem Wissenszuwachs im Projekt wachsen, also um neue Anforderungen ergänzt werden. Diese werden regelmäßig priorisiert und bewertet. Hoch priorisierte Features werden in das Sprint Backlog übernommen. Das Sprint Backlog enthält alle Aufgaben, die das Team im anstehenden Sprint bearbeitet. Der Erfüllungsgrad der Aufgaben wird täglich in einem sog. Burndown-Chart dargestellt.

Der Sprint selbst ist das zentrale Element des Prozessmodells. Er kennzeichnet eine Iteration in Form einer Time Box. Jeden Tag soll ein sog. Daily Scrum durchgeführt werden. Dieses Ritual dient dazu, dass jeder im Projektteam berichtet, was sein aktueller Arbeitsstatus ist, was er bis zum nächsten Daily Scrum bearbeiten will und wo er möglicherweise Probleme im Projekt hat, die der Scrum Master lösen muss. Ziel ist es, dass möglichst jeder im Projekt über den aktuellen Status informiert ist. Nach jedem Sprint soll eine lauffähige Version der Software vorliegen, die durch den Kunden durch ein Review geprüft wird. Änderungsforderungen, die der Kunde im Rahmen des Reviews aufstellt, werden wieder in das Product Backlog eingepflegt. Ebenfalls am Ende eines Sprints ist das Projektteam dazu aufgerufen, in eine Retrospektive zu gehen und den zurückliegenden Projektabschnitt zu bewerten. Die Erfahrungen werden im nächsten Sprint berücksichtigt, bzw. an den Scrum Master zurückgemeldet, wenn Probleme zu beseitigen sind.

Scrum ist eine agile Methode, die hohe Anforderungen sowohl an die Organisation als auch an die Projektteams und die einzelnen Mitarbeiter stellt. Die Organisation muss einen hohen Reifegrad besitzen und insbesondere Techniken wie automatische Testverfahren und Refaktorierung beherrschen. Die Mitarbeiter müssen in modernen Programmiertechniken ausgebildet sein, um in den kurzen Iterationen des Projekts auch Ergebnisse produzieren zu können. Weiterhin ist Scrum von einem sozial ausgewogenen Team abhängig. Dominante Einzelcharaktere stören das selbstorganisierende Team und wirken negativ auf die Produktivität.



## Extreme Programming

The first Extreme Programming project was started March 6, 1996. Extreme Programming is one of several popular Agile Processes. It has already been proven to be very successful at many companies of all different sizes and industries worldwide.

Extreme Programming is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it. Extreme Programming empowers your developers to confidently respond to changing customer requirements, even late in the life cycle.

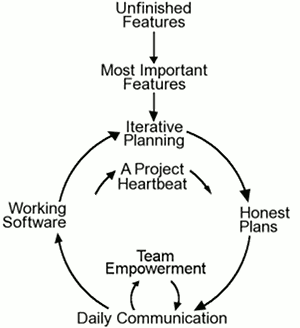
Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.

Extreme Programming improves a software project in five essential ways: communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. Every small success deepens their respect for the unique contributions of each and every team member. With this foundation Extreme Programmers are able to courageously respond to changing requirements and technology.

The most surprising aspect of Extreme Programming is its simple rules. Extreme Programming is a lot like a jig sawpuzzle. There are many small pieces. Individually the pieces make no sense, but when combined together a complete picture can be seen. The rules may seem awkward and perhaps even naive at first but are based on sound values and principles.

Our rules set expectations between team members but are not the end goal themselves. You will come to realize these rules define an environment that promotes team collaboration and empowerment, that is your goal. Once achieved productive teamwork will continue even as rules are changed to fit your company's specific needs.

This flow chart shows how Extreme Programming's rules work together. Customers enjoy being partners in the software process, developers actively contribute regardless of experience level, and managers concentrate on communication and relationships. Unproductive activities have been trimmed to reduce costs and frustration of everyone involved.



## Kanban

Entwickelt wurde Kanban bereits in den Vierzigerjahren des vergangenen Jahrhunderts von dem Automobilkonzern Toyota, um Lagerbestände zu reduzieren und die Fertigungsprozesse in einen gleichmäßigeren Rhythmus zu bringen. Als Kanban (japanisch kan = Signal, ban = Karte) wird die horizontale, japanische Adaption der klassischen To-Do-Liste bezeichnet. Durch Kanban lässt sich der Workflow besser planen, überwachen und steuern. Ziel des Kanban-Boards ist es, Projektabläufe und Aufgaben zu visualisieren – in übersichtlichen Spalten mit einzelnen Einträgen, die der Reihe nach abgearbeitet werden können. So können frühzeitig eventuelle Engpässe erfasst und behoben und die Wertschöpfungskette kostenoptimal gelenkt werden.

### Kanban Board

Das Ganze funktioniert ebenfalls recht simpel und in Form einer Wandtafel. Die Kanban Tafel wird in der Grundversion in drei Spalten unterteilt, auf die dann jeweils Haftnotizen oder Karteikarten mit der jeweiligen Aufgaben kommen.

Als typische Drei-Felder-Wirtschaft gelten diese Spaltenbezeichnungen:

* Aufgabe (To Do) Hier werden zunächst alle erforderlichen Prozesse und Aufgaben erfasst. Dies ähnelt der klassischen To Do Liste.
* Bearbeitung (Doing) Sobald erste Arbeiten ausgeführt werden und die Umsetzung begonnen hat, wandert die jeweilige Karte in diesen Bereich.
* Erledigt (Done) Hier hin gelangt ein Haftzettel erst dann, wenn die darauf stehende Aufgaben wirklich vollständig abgearbeitet wurde.

So lässt sich vor allem die mittlere Spalte um diverse Analyse- oder Zwischenschritte erweitern, die Karten (beziehungsweise Aufgaben) wiederum lassen sich mit Anmerkungen versehen, sodass der gesamte Prozess sowie einzelne Zwischenstände für alle Teammitglieder ersichtlich und transparenter werden.

Überdies können Sie einzelnen Teammitgliedern unterschiedliche Post-it-Farben zuweisen und so bei einem gemeinsamen Projekt relativ schnell am Kanban-Board erkennen…

* Wer gerade was macht
* Wer wo steht
* Wer bummelt oder vielleicht Unterstützung braucht
* Wer Zeit hat, zu helfen

Zusätzlich hilft das Kanban Board dabei, die Dauer von Prozessen zu erkennen, so dass auch Prognosen für Kunden möglich sind. Bis man eine durchschnittliche Durchlaufzeit von Prozessen allerdings ermitteln kann, brauchen Sie Erfahrungswerte aus den vorangegangenen Wochen, das heißt, es muss sich der Gebrauch eines Kanban Boards erst eingespielt haben.

Mit wachsender Erfahrung funktioniert die Umsetzung nicht nur besser, das Kanban Board kann auch individuell entwickelt und an die Bedürfnisse angepasst werden. Die grundsätzliche Idee des Kanban bleibt dabei erhalten, es können jedoch Anpassungen vorgenommen werden. So kann sich beispielsweise herausstellen, dass auf den einzelnen Notizen zusätzliche Informationen festgehalten werden müssen; oder dass die Priorität der einzelnen Aufgaben farblich dargestellt werden muss; oder dass Symbole auf die verschiedenen Projekte hinweisen, zu denen die einzelnen Tasks gehören.

### Prinzipen und Praktiken

David Anderson, einer der Begründer agiler Softwareentwicklung, beschrieb vier Grundprinzipien und sechs Praktiken, die Unternehmen bei der Anwendung von Kanban in ihre Arbeitsweise einbauen.

Grundprinzipien:

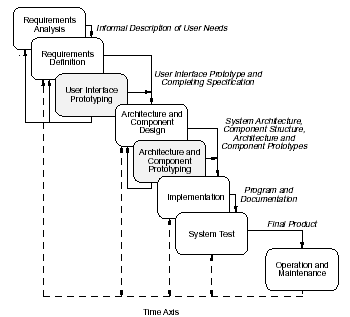
* Start with what you do now: Starte mit dem, was du jetzt machst
* Agree to pursue incremental, evolutionary change: Verfolge inkrementelle, evolutionäre Veränderung
* Respect the current process, roles, responsibilities and titles: Respektiere gegenwärtige Prozesse, Rollen, Verantwortlichkeiten und Ansprüche
* Encourage acts of leadership at all levels in your organization – from individual contributor to senior management: Fördere Leadership auf allen Ebenen in der Organisation – vom einzelnen Mitarbeiter bis zur Geschäftsleitung

### Prinzipien und Praktiken

1. Visualisierung 🡪 Prozessschritte werden mit Post-it‘s am Kanban-Board dargestellt
2. Begrenzung 🡪 Um einen gleichmäßigen Workflow zu gewährleisten, wird die Anzahl der Tickets (Work in Progress – WiP) begrenzt, die zur selben Zeit an einer Station bearbeitet werden dürfen. Dieses Vorgehen wird Pull-Prinzip genannt: jede Station holt ihre Arbeit bei der Vorgängerstation ab, anstatt fertige Arbeit einfach an die nächste Station weiterzureichen
3. Steuerung 🡪 einzelne Bereiche werden überprüft, um eventuelle Verbesserungen festzustellen.
4. Verdeutlichung 🡪 Alle Beteiligten müssen von denselben Annahmen und Gesetzmäßigkeiten ausgehen, deswegen gibt es Regeln:
   * Begriffsdefinition
   * Bedeutung der jeweiligen Spalte
5. Rückmeldung 🡪 Kanban ist ein flexibles Modell, daher werden beständig Rückmeldungen und Überprüfungen durchgeführt. Durch diese Feedbacks können bottlenecks (Stauungen) erkannt und behoben werden.
6. Verbesserung 🡪 Feedback führt zu Verbesserungen

# Prototyping

Während der Softwareentwicklung werden verschiedene Prototypen für verschiedene Probleme benötigt. Prototyping ist im Softwareentwicklungsprozess wichtig, weil: Ein Prototyp ist ein funktionierendes Modell des Systems und enthält bereits gewisse Aspekte des Systems. Sie leisten wichtige Unterstützung bei der Problemfindung, beim Erfassen von Schwierigkeiten und beim Treffen von Entscheidungen (wenn nötig unterstützt durch das „Pflichtenheft“). Jeder Prototyp stellt eine Basis für nachfolgende Prototypen oder für das gesamte System dar.



**Verschiedene Arten von Prototypen**

Grundsätzlich unterscheiden wir zwischen vollständigen und unvollständigen, sowie zwischen Wegwerf- und Wiederverwendbaren Prototypen.

*Vollständig* heißt, dass alle wesentlichen Funktionen des geplanten Systems vollständig verfügbar sind. Die bei der Herstellung und beim Einsatz gemachten Erfahrungen bilden die Grundlage für die endgültige Systemspezifikation (für Softwaresysteme kaum verwendet).

*Unvollständig* heißt, dass der Prototyp die Brauchbarkeit und Machbarkeit einzelner Aspekte des geplanten Systems untersucht.

*Wegwerfprototyp*: Die Implementierung des Prototyps wird bei der Implementierung des Zielsystems nicht weiterverwendet. Der Prototyp dient nur als Modell.

*Wiederverwendbarer Prototyp***:** Wesentliche Teile des Prototyps werden bei der Implementierung des Zielsystems übernommen

* „prototyp proper“: Darunter verstehen wir ein vorausschauendes Softwaresystem das parallel zum Systemmodell gemacht wird. Diese Art von Prototyp wird normalerweise entwickelt, um spezielle Aspekte des Userinterfaces oder Teile der Funktionalität anschaulich zu machen und hilft Probleme damit zu erkennen. Es ist besser mehrere kleine Prototypen zu bauen, die mit mehreren Problemen umgehen als einen einzigen immer wieder auszubauen, weil die meisten Probleme weder klar definiert noch homogen sind.
* „breadboard“: Darunter versteht man einen Prototypen der dem Entwicklerteam hilft, konstruktionsbezogene Fragen zu klären. Ein breadboard stammt ab vom Systemmodell oder der Software Spezifikation. Die Benutzer sind normalerweise ausgeschlossen bei der Entwicklung. Die Herstellung eines breadboard hat verschiedene Anforderungen mit einem prototyp proper gemeinsam: Die modellierten Aspekte müssen, aus einer technischen Sicht, vergleichbar mit dem künftigen Anwendungssystem sein. Abhängig vom Problem kann diese Vergleichbarkeit mit der Systemarchitektur oder der Funktionalität verknüpft sein.
* „pilot system“: Wenn ein Prototyp nicht nur für das Testen einer Idee oder für Veranschaulichungszwecke genutzt wird sondern als Kern eines Anwendungssystem gilt, spricht man von einem Pilot System. In diesem Fall hört die klare Unterscheidung zwischen Anwendungssystem und Prototyp auf. Nach Erreichen einer gewissen Komplexität wird der Prototyp in Form eines Pilotsystems implementiert. Während die Benutzeroberfläche definiert wird, sollten die verschiedenen Pilotsysteme nur mit den Benutzerprioritäten arbeiten. Aus technischer Sicht ist ein Pilotsystem vom Design her viel umfangreicher als ein prototyp proper oder ein breadboard. Zusammenfassend ist zu sagen, dass ein Pilotsystem für die Anwendungsumgebung eingesetzt wird und nicht nur für Laborzwecke.

Als Erklärung: Prototyping kann verwendet werden, um eine Vielzahl von Aktivitäten in einem Entwicklungsprojekt zu unterstützen.

**Ziele des Prototyping:**

Wenn man Prototypen konstruiert, konzentriert man sich auf die großen Probleme, die in der typischen Softwareentwicklung auftreten und die mit Prototyping gelöst werden können. Hierbei unterscheidet man zwischen verschiedenen Arten von Prototyping, die jeweils auf verschiedene Ziele ausgelegt sind:

*Das explorative Prototyping:*

Das Ziel ist eine möglichst vollständige Systemspezifikation. Der Zweck ist den Entwicklern einen Einblick in den Anwendungsbereich zu ermöglichen und Lösungsansätze mit den Anwendern zu diskutieren. Diese Methode wird benutzt, wenn das Problem noch unklar ist. Lösungsideen werden

hier eingesetzt um Benutzer- und Managementanforderungen zu klären in Hinblick auf das künftige System. Wichtig ist hierbei die Abdeckung einer Vielzahl von Designmöglichkeiten, sodass nicht nur eine Lösungsmöglichkeit für eine spezifische Anforderung entsteht. Der Entwickler gewinnt hierdurch Einsicht in das ganze Gebiet der Anwendung und ihre Probleme („prototyp proper“). Das explorative Prototyping bietet Vorteile, wenn Entwickler und Anwender verschiedenen Organisationen angehören und erfüllen hier die Funktion eines „Behälters“, der dem Management der Benutzerorganisation zeigt wie das System im Ganzen aussehen könnte. Die Realisierung des Prototyps wird von Anwendern und Entwicklern gemeinsam vorgenommen. Wichtig ist dabei nicht die Qualität des Prototyps, sondern seine Funktionalität, die leichte Veränderbarkeit und auch die Kürze der Entwicklungszeit.

*Das experimentelle Prototyping:*

Das experimentelle Prototyping konzentriert sich auf die technische Implementierung eines Entwicklungszieles. Der Benutzer kann seine Ideen über den Typ der Computerunterstützung, die er braucht, spezifizieren, während die Entwickler mit einer Basis für die voraussichtliche Nutzbarkeit und Kompatibilität eines Anwendungssystems versorgt werden. Ein wichtiger Aspekt hier ist die

Kommunikation zwischen Benutzer und Entwickler über technische Probleme und Fragen, welche die Struktur der Software betreffen. Diese Art von Prototypen helfen also dem Entwickler, technische Probleme schnell zu klären (meist verwendet man hierfür „bread boards“).

Für die Qualität gilt das Gleiche wie beim explorativen Prototyping. Realisiert werden die Prototypen für die Durchführung von Experimenten in der Hauptsache von den Entwicklern. Das experimentelle Prototyping ist auch eine Technik zur Unterstützung beim System- und Komponentendesign.

*Das evolutionäre Prototyping:*

Ziel ist eine inkrementelle Systementwicklung, das heißt eine schrittweise aufbauende Entwicklung eines Prototyps für die von vorneherein klare Benutzeranforderung. Hierbei wird das Prototyping nicht auf ein einzelnes Entwicklungsprojekt angewandt, sondern ist ein stetiger Prozess um Anwendungssysteme an sich schnell ändernde Anforderungen in der Organisation anzupassen. Das heißt, dass die Softwareentwicklung nicht als selbstbezogenes Projekt, sondern als stetiger Prozess,

der die Anwendung begleitet, gesehen wird. Das evolutionäre Prototyping ist daher eng verknüpft mit den Prinzipien von evolutionären Systementwicklungen. Dadurch ändert sich auch die Rolle der Entwickler, die nicht mehr alleine an einem nach außen abgeschlossenen Projekt arbeiten, sondern vielmehr technische Berater werden, die in Kooperation mit dem Benutzer arbeiten, um das System zu verbessern. Die Systementwicklung verliert den Charakter eines abgeschlossenen Projekts und

wird ein Prozess, der die Anwendung ständig begleitet. Es lässt sich nicht mehr definitiv zwischen Prototyp und Produkt unterscheiden. Man verwendet den Begriff Prototyping aber trotzdem, und zwar deswegen, weil die ersten Versionen sicher noch nicht als Produkt dienen können. Die Prototypen werden nicht simuliert und auch nicht weggeworfen, sondern Schritt für Schritt zum Produkt ausgebaut.

**Design von Prototypen**

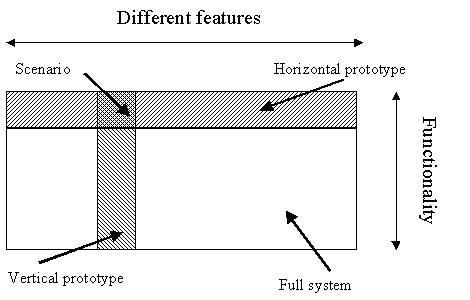
Es gibt verschiedene Designs von Prototypen, die im Hinblick auf ihren Verwendungszweck ausgewählt werden. Man spricht vom

*Horizontales Prototyping*

Wenn nur spezielle Teile bzw. Bereiche des Systems implementiert werden. Zum Beispiel: die Benutzeroberfläche mit den Grafiken, Menüs, etc. oder funktionelle Bereiche wie Datenbankenaktionen, Schnittstellen. Wir verstehen unter horizontalen Prototyping also das Prototyping von Mensch-Computer-Schnittstellen. Der Zweck ist es, dem Benutzer einen Einblick in die Bedienung des Systems zu gewähren. Im Gegensatz steht das

*Vertikales Prototyping*

Beim vertikalen Prototyping wird ein Teil des Gesamtsystems ausgewählt und komplett implementiert. Ziel ist den Entwicklern des Systems die Möglichkeit zu geben, einzelne Komponenten und Funktionen davon zu testen. Wie das Gesamtsystem implementiert wird und ob es funktioniert, bleibt aber weiterhin offen. Meist verwendet man hier „Pilotsysteme“ .



**Prototyping Techniken**

System Prototyping Techniken müssen die schnelle Entwicklung eines Prototypen ermöglichen können. Da die Kosten für Mitarbeiter die höchsten Ausgaben in der Softwareentwicklung sind, ist mit der schnellen Entwicklung gemeint das die Kosten für den Prototypen minimiert werden. Es soll auch ermöglicht werden, dass das Feedback des Benutzers schon früh in den Softwareprozess einlaufen kann. Daher gibt es eine Anzahl von Techniken, die sich im System Prototyping bewährt haben:

*Ausführbare Spezifikationssprachen*

Im Wesentlichen geht es um die formale Spezifikation von Softwaresystemen (formale Spezifikation ist ein mathematisches Systemmodell, das auch ausführbar sein kann). Dabei muss man auch Generalität von der Spezifikation vermindern, dafür wurde eine Anzahl von ausführbaren Spezifikationssprachen entwickelt.

*Very-High-Level-Sprachen*

Very-High-Level-Sprachen sind Programmiersprachen mit dynamischen Datenstrukturen. Zum Beispiel LISP (basiert auf Listen) Prolog (basiert auf Logik) Smalltalk (basiert auf Objekten) APL (basiert auf Vektoren) SETL (basiert auf Sets). Dies sind nützliche Prototyping Sprachen, weil die dynamischen Features schnelle Systementwicklungen möglich machen. Very-High-Level-Sprachen werden normalerweise nicht für umfangreiche Systementwicklungen benützt, da sie ein dementsprechend großes Laufzeit-Unterstützungs-System benötigen. Dieses System erhöht den Speicherbedarf und reduziert die Ausführungszeiten des Programms. Da jedoch Leistungsanforderungen in der Prototypentwicklung in der Regel keine bis geringe Rolle spielen, ist dies nicht unbedingt immer ein Nachteil.

*4.-Generationssprachen (anwendungsorientierte High-Level-Sprachen)*

4GLs sind erfolgreich, weil sie in Bezug auf Datenprozessabläufe viel gemeinsam haben. Viele Anwendungen sind Reportaktivitäten, die Informationen in einer Datenbank benutzen. Weil eine typische 4GLs auf einer Datenbank basiert und eine Datenbank Query Language anbietet, beinhaltet sie auch ein Report Package und ein Package für das Grafik Design. Eine 4GL arbeitet mit Software Reuse (Wiederverwertung) in der die Routinen für den Zugriff auf eine Datenbank und für das Produzieren von Reporten schon gegeben ist. Der Programmierer muss nur noch beschreiben, wie diese Routinen kontrolliert werden. Das schlagende Argument der Verkäufer dieser Produkte ist, dass die Systemerhaltung einfacher und billiger ist, weil die Zeit für die Entwicklung einer Anwendung geringer ist. Die Systeme sind normalerweise auch viel kleiner als mit anderen Sprachen. Es ist daher in manchen Fällen, vor allem für Datenverarbeitungssysteme, kosteneffektiv diese Entwicklungsart zu verwenden.

*Zusammensetzen von wiederverwertbaren Komponenten*

Es leuchtet ein, dass die Existenz einer Bibliothek von wiederverwertbaren Komponenten die Entwicklungszeit eines Systems reduziert. Wenn diese Komponenten genug generalisiert sind und ein Mechanismus zur Verfügung steht, der diese Komponenten in das System integriert, kann ein Prototyp sehr schnell erstellt werden. Das beste Beispiel dafür ist die Shell – Programmiersprache in Unix. Dieses Shell ist eine Programmiersprache, die Schleifen und Entscheidungskonstrukte enthält. Sie bietet die Möglichkeit, Anweisungen zu kombinieren, die mit Daten und Strings arbeiten. Dennoch sind dieser Art von Prototyping Grenzen gesetzt, weil die Funktionen der einzelnen Komponenten meist zu sehr oder zu wenig generalisiert sind, um sie effektiv mit anderen Komponenten zu verbinden. Außerdem ist das Prototyping von User-Interfaces mit diesem Shell nicht optimal möglich wegen dem einfachen I/O-Modell, das Unix verwendet. Der Vorteil ist aber, dass diese Art von Prototyping wiederum mit anderen Arten kombiniert werden kann (Very-HighLevel-Sprachen oder 4GLs).

Daher kann man sagen, dass die Mächtigkeit von Programmiersprachen wie Smalltalk oder LISP beim Prototyping mehr von der verfügbaren Bibliothek der wiederverwertbaren Komponenten abhängt als von ihren technischen Voraussetzungen und ihren Bedienungsmerkmalen.