

Introduction to Software Engineering

Felix Leidl

28. Februar 2024

Inhaltsverzeichnis

Software processes	3
Software specification	3
Requirements election and analysis	3
Requirements specification	3
Requirements validation	3
Software design and implementation	3
Architectural design	3
Database design	3
Interface design	3
Component selection and design	4
Software verification and validation	4
Component testing	4
System testing	4
Customer testing	4
Software evolution	4
Change anticipation	4
Change tolerance	4
Software development life circle	5
Life cycle phases	5
Software Process Models	5
Wasserfallmodell	6
Improved Waterfall model	7
V-Modell	8
Wiederverwendungsansatz	10
Agiles Modell	11
Change Management	11
Prototyp	11
Schrittweise Veröffentlichung	12

Agile software development	12
Agile manifesto	13
Rational Unified Process (RUP)	13
Vier Phasen	13
Disziplinen	14
Kanban	15
Practices	15
Extreme programming	16
Core values	16
Vier Aktivitäten	16
Planning	16
Design	17
Implementation	18
Testing	18
Timeline overview	18
Requirements engineering	19
System modeling	19
Architectural design	19
Design patterns	19
Implementation	19
Software testing	19
Software evolution	19
Software project management	19
Software engineering in machine learning	19

Software processes

Software specification

Requirements election and analysis

- Beobachtung des existierenden Systems
- Absprache mit Nutzern und Entwicklern
- Anforderungsanalyse
- Entwicklung von Modellen und Prototypen

Requirements specification

- Anforderungen formulieren und dokumentieren
- Nutzeranforderungen (abstrakt)
- Systemanforderungen (detailliert)

Requirements validation

- Umsetzbarkeit
- Konsistenz
- Vollständigkeit
- Fehlerkorrektur

Software design and implementation

Architectural design

- Systemstruktur
- Hauptsächliche Strukturen und Beziehungen
- Vertrieb

Database design

- Datenstrukturen
- Darstellung in Datenbanken

Interface design

- Eindeutige Interface-Spezifikationen
- Kommunikation zwischen Komponenten, ohne Kenntnis der Implementation

Component selection and design

- Suche nach wiederverwendbaren Komponenten
- Definiere Veränderungen bei wiederverwendeten Komponenten
- Entwerfe neue Komponenten

Software verification and validation

Component testing

- Komponenten durch Entwickler testen
- Individuelle Tests, ohne andere Komponenten

System testing

- Komplettes System ist getestet
- Fehler von unvorhergesehenen Verwendungen und Interfaces sind behoben
- Beweise, dass das System die Anforderungen erfüllt

Customer testing

- Letzte Hürde vor Veröffentlichung
- System ist von Nutzer mit echten Daten verwendet worden
- Anforderungsprobleme müssen behoben werden

Software evolution

Es gibt zwei Wege mit Veränderung umzugehen:

Change anticipation

- Mögliche Veränderungen vorhersehen
- Neustart minimieren
- z.B. erst Prototyp erstellen, dann das ganze Produkt

Change tolerance

- Design berücksichtigt Veränderungen am System
- Normalerweise durch schrittweise Entwicklung
- Ein kleiner Schritt ist genug um eine Veränderung anzunehmen

Software development life circle

Life cycle phases

1. Initialisierung, Konzept entwickeln, vorläufige Planung, Anforderungsanalyse (→ Spezifikation)
2. Design: High-level & Low-level Design
3. Implementation
4. Validierung & Verifikation
5. Vertrieb: Veröffentlichung der Anwendung
6. Erhaltung (→ Evolution)
7. Beginne von vorne
8. Bis Absetzung: Planen der Entfernung der Software (aufräumen & archivieren)

Software Process Models

Ein Prozess-Modell ist eine abstrakte Repräsentation der Aktivitäten während des Softwareentwicklungsprozesses um:

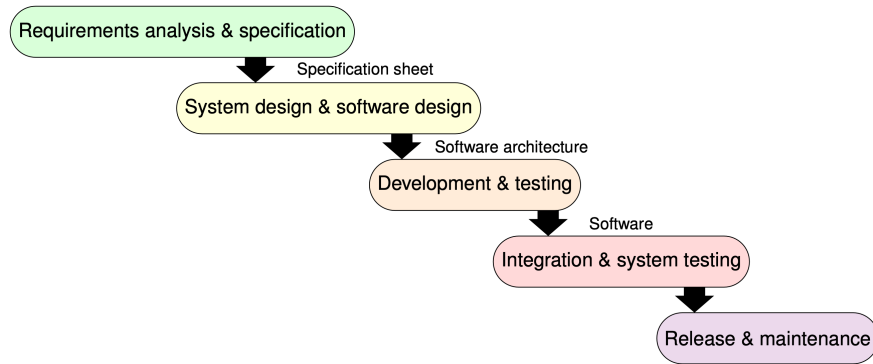
- Abläufe zu definieren
- die Ablaufordnung zu spezifizieren
- Phasen zu determinieren: Abläufe, Ziele, Rollen und Methoden

Die Nutzung von Prozess-Modellen führt zu:

- einer Richtlinie für die Systementwicklung
- einer einheitlichen Ansicht gegenüber logischer und temporärer Planung
- besserer Planung
- Unabhängigkeit von einzelnen Personen
- möglichen Zertifikaten
- früherer Erkennung von Fehlern durch Tests

Wasserfallmodell

Tabelle 1: Waterfall model



Requirements analyses & specification: Projektmanagement beginnt, Probleme und Spezifikationen werden zusammengestellt, Anforderungen definiert und dokumentiert

System & softwaredesign: Entwürfe, Modelle und die Softwarearchitektur werden entwickelt

Development & testing: Software entwickeln und durch Unit-Tests verifizieren

integration & systemtests: Software Komponenten kombinieren und das Gesamtsystem testen

Release & maintenance: System installieren, Fehler korrigieren, Software an Altern hindern, neue Anforderungen bearbeiten

Pros:

- Linearer Prozess
- Intuitiv
- Einfach verständlich
- Top-Down
- Planbar
- Nicht-Unterbrechbar

Cons:

- Feste Phasen
- Frühe Festlegung
- Keine Wiederholung
- Kein Einbeziehen neuer Anforderungen
- Oft unpraktisch

Verwendung:

- Anforderungen sind einfach zu definieren und ändern sich nicht
- Projekt, Budget und Prozess sind vorhersagbar
- Strikter Prozess ist notwendig

Improved Waterfall model

Tabelle 2: Iterative Waterfall model

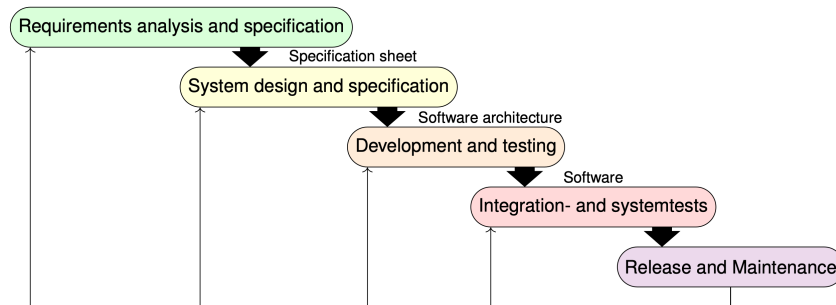
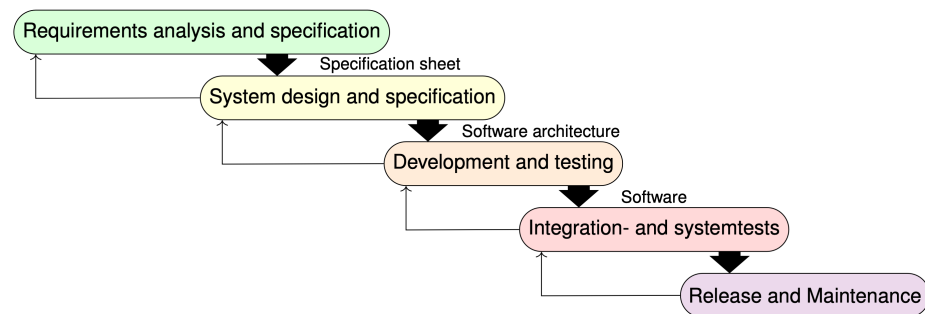


Tabelle 3: Incremental Waterfall model



Pros:

- Linearer Prozess
- Intuitiv
- Einfach zu Verstehen
- Top-Down
- Planbar
- Nicht-Unterbrechbar
- Wiederholbar

Cons:

- Gefixte Phasen
- Frühe Festlegung, aber neue Anforderungen können integriert werden
- Veränderte Anforderungen können zu hohen Kosten führen
- Struktur tendiert abzubauen

Verwendung:

- Anforderungen sind einfach definiert

- Nur kleine Änderungen können erscheinen und sind im Budget mit eingerechnet
- Projekt, Budget und Prozess sind vorhersagbar
- Strikter, aber leicht flexibler Prozess ist notwendig

V-Modell

Tabelle 4: V-model

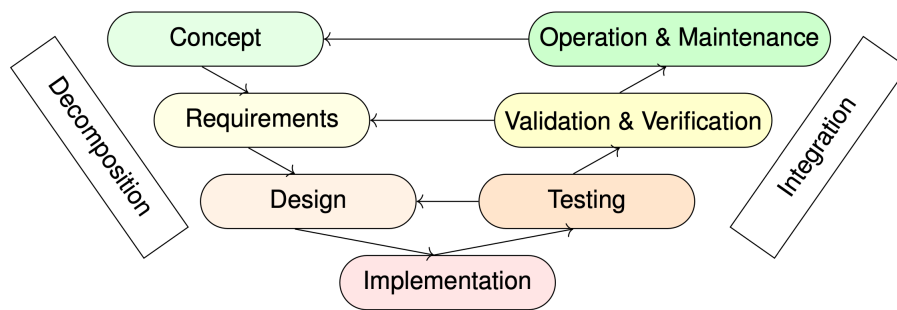
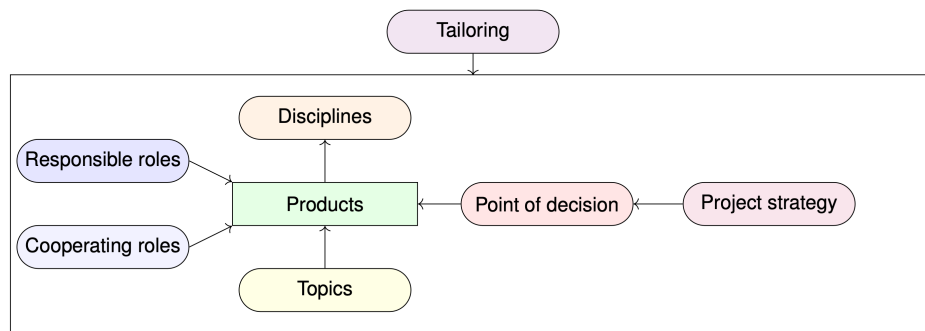
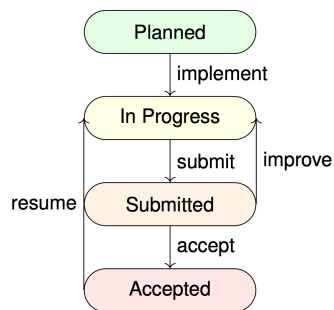


Tabelle 5: V-model XT



Jedes Produkt geht durch vordefinierte Zustände



Pros:

- Bei großen und komplexen Systemen anwendbar
- detaillierte Spezifikationen, Rollendefinitionen und Ergebnisstrukturen
- Qualitätsorientiert

Cons:

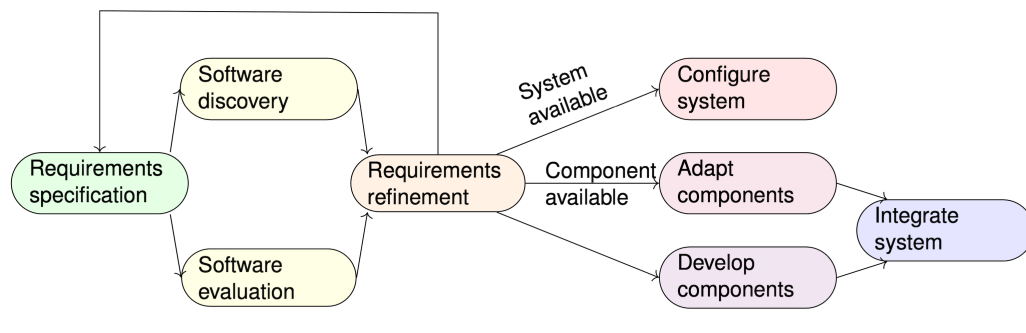
- Großer Overhead bei kleinen Projekten
- Testphase beginnt relativ spät
- Strikte Phasen
- Teilnehmer müssen geschult sein, um Modell zu folgen

Verwendung:

- Softwareentwicklung in Behörden
- Sicherheitsrelevante Projekte

Wiederverwendungsansatz

Tabelle 6: Reuse-oriented approach



Pros:

- Reduziert Kosten und Risiken
- Schnellere Verteilung

Cons:

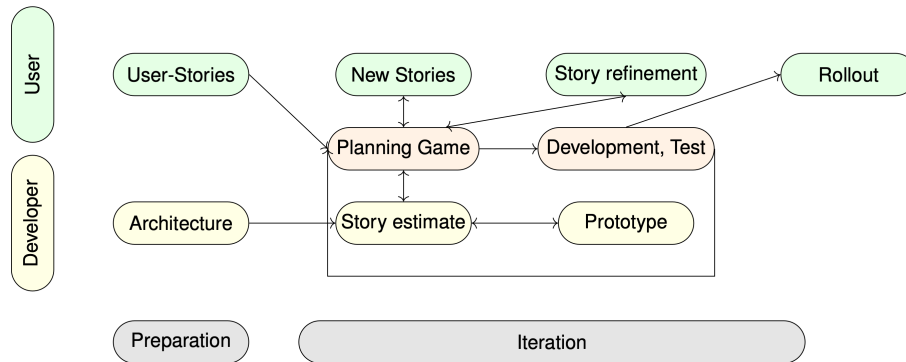
- Kompromisse in den Anforderungen
- System könnte echte Nutzerbedürfnisse nicht erfüllen
- Kein oder limitierte Kontrolle über Systemevolution

Verwendung:

- Webanwendungen
- Collection of objects or packages
- Konfigurierbare stand-alone Softwaresysteme

Agiles Modell

Tabelle 7: Agile model



Pros:

- Begrenzter bürokratischer Aufwand
- Flexible Rollen
- so wenig Dokumentation wie nötig
- besseres Kosten/Nutzen Verhältnis
- Bessere Code-Qualität

Cons:

- Ganzes Team muss den Regeln folgen
- Projektergebnis nicht vorhersehbar

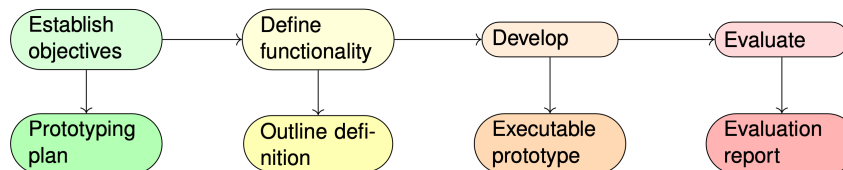
Verwendung:

- Große, komplexe sowie kleine Systeme
- Projekte die Prototypen erfordern

Change Management

Prototyp

Tabelle 8: Prototyping



Pros:

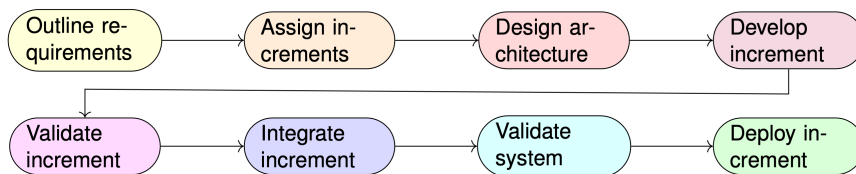
- Kunde gibt Priorität der Anwendung vor
- Software kann sofort verwendet werden
- Kunde erlangt Erfahrung mit dem System
- Kunde kann Anforderungen für spätere Schritte abgeben
- Veränderungen sind einfach umzusetzen
- Die wichtigsten Systemteile werden am häufigsten getestet

Cons:

- Softwarebasis kann ohne detaillierte Anforderungen nicht identifiziert werden
- Kann mit organisatorischen Strukturen in Konflikt geraten (z.B. Verträge)
- Unbrauchbar um existierende Systeme zu ersetzen

Schrittweise Veröffentlichung

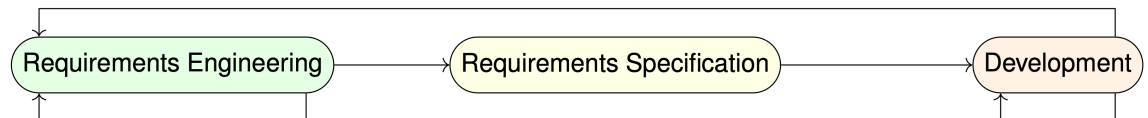
Tabelle 9: Incremental delivery



Agile software development

Tabelle 10: diff. between plan-driven and agile

Plan-driven:



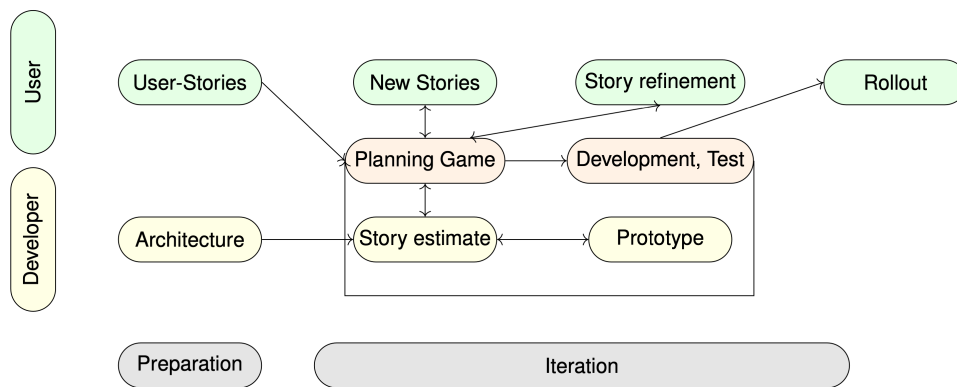
Agile:



Agile manifesto

1. Individuen und Interaktionen über Prozessen und Werkzeugen
2. Funktionierende Software über akribischer Dokumentation
3. Zusammenarbeit mit dem Kunden über Vertragsverhandlungen
4. Auf Veränderungen eingehen über Plan folgen

Tabelle 11: Generic model



Rational Unified Process (RUP)

Vier Phasen

- Inception: Beginn des Projekts, Business-Modell, grundsätzliche Anforderungen und Bedingungen werden definiert
- Elaboration: Anforderungen spezifizieren, Architektur, Design und Iterationen
- Construction: Komponenten werden entwickelt und getestet
- Transition: Erschaffung von „artifacts“ und Konfiguration, Veröffentlichung

Jede Phase kann mehrfach Wiederholt werden, wird von einem Meilenstein abgeschlossen, liefert „artifacts“, welche Ergebnisse früher spezifizierter Aktivitäten sind und wird Wiederholt, wenn die „artifacts“ nicht ausgeliefert werden oder die Standards nicht erreichen

Disziplinen

Engineering Workflow

- Business modeling:
 - Allgemeines Verständnis aller „stakeholders“ der Software
 - z.B. Komponenten-Diagramme, Use-Case-Diagramme, Klassen-Diagramme
- Requirements: Detaillierte Spezifikation des initialen Use-Case und Businessmodelle
- Analysis & Design:
 - Architektur des Systems wird aus Anforderungen erarbeitet
 - Architektur-, Design- und Testdokumente
 - Klassen- und Zusammenhangsdiagramme
- Implementation: Definiert, wie Komponenten implementiert, getestet und integriert werden
- Test:
 - Beginnt früh im Projekt
 - Erhöht Verständnis des Systems
 - Wird ausgeführt, sobald Komponenten, Subsysteme und System verfügbar ist
- Deployment: Finalisieren und veröffentlichen des Produkts

Supporting Workflow

- Configuration & change management:
 - Organisiert Konfigurations- und Versionsmanagement
 - Versionsmanagement der Dokumentation
- Project management:
 - Planung und Koordination des Projekts
 - Steuert Ressourcen, Qualität und Quantität
 - Entscheidet, ob zusätzliche Wiederholungen notwendig sind
- Environment: Definiert verfügbare Ressourcen für die Entwicklungsteams

Pros:

- Für Großprojekte geeignet
- Definiert Produkt, Rollen und Aktivitäten
- Umfangreiche Nutzung der UML, um echte Szenarien zu mo-

modellieren

Cons:

- Komplex
- Nicht flexibel
- Große Anzahl an Dokumenten

Kanban

Kanban verwendet ein visuelles, pull-basiertes System, um den Flow zu optimieren

Practices

- Workflow definieren und visualisieren
- Aktiv Pakete im Workflow managen
- Workflow verbessern

Tabelle 12: Kanban board



Damit Kanban funktioniert muss ein einheitliches Verständnis des Workflows existieren (**Definition of Workflow**)

- „Work items“ definieren - individuelle Einheiten, welche durch den Workflow wandern
- Definieren, wann „work items“ beginnen und enden
- Definieren, durch welche Zustände „work items“ gehen
- Definieren, wie „**Work In Progress**“ kontrolliert wird

- Explizite Richtlinien für die „Wanderung“ durch die Zustände
- Ein „Service Level Expectation“: Erwartete Durchlaufzeit, jedes Items

Der Mindestsatz der Durchlaufmetriken besteht aus:

- WIP: Anzahl der Items, die begonnen, aber nicht abgeschlossen wurden
- Throughput: Anzahl der abgeschlossenen Items pro Zeiteinheit
- Work Item Age: Deltazeit seit Bearbeitungsbeginn
- Cycle Time: Deltazeit zwischen Bearbeitungsbeginn und Abschluss

Extreme programming

Core values

- Kommunikation
- Einfachheit (Simplicity)
- Feedback
- Mut (Rewrites und ehrliche Kundenkommunikation)
- Respekt

Vier Aktivitäten

- Planning
- Design
- Implementation
- Testing

Planning

- **User Stories :**
 - Kurz beschreiben, was passieren soll
 - Wird für Risikoanalysen, Aufwandsschätzung und Wiederholungsplanung verwendet
 - Von Firmenseite entwickelt, mit Hilfe der Entwickler
- **Planning Game :**
 - Endet in einem Veröffentlichungsplan
 - 4 Variablen: Umfang, Ressourcen, Zeit und Qualität

- User stories werden beauftragt, um ein Projektplan zu erstellen, der jeden zufrieden stimmt
- Ein Veröffentlichungsplan besteht aus 80 ± 20 user stories
- **Iteration Planning :**
 - Iterationen werden mit dem Kunden geplant
 - User stories werden ausgewählt
 - Implementierungsaufgaben und Testfälle werden den User stories entnommen
- **Small releases:** Schnelleres Feedback der Kunden, jedoch keine unnötigen Veröffentlichungen
- **Stand – Up Meeting :** Kurzes Meeting, um Status, Probleme und Lösungen zu besprechen
- **Measuring project progress**
- **Move People Around**
- **Fix XP when it breaks**

Design

- **Simplicity:** Finales Design erst kurz vor Ende und Implementation nur wenn nötig
- **Spike solutions:** Aus Prototypen für Implementation Schlüsse ziehen
- **Refactoring:** Computer Aided Software Engineering
- **CRC cards:**
 - Class, Responsibility and Collaboration
 - Beschreibe eine Klasse mit maximal 4 Sätzen
 - Ein Satz Klassen pro Wiederholung
 - Die Verbindung der Klassen bildet ein UML-Diagramm

Front:	Class Name Superclass	Back:	Operations
	Responsibility		Attributes
	Collaboration		

Implementation

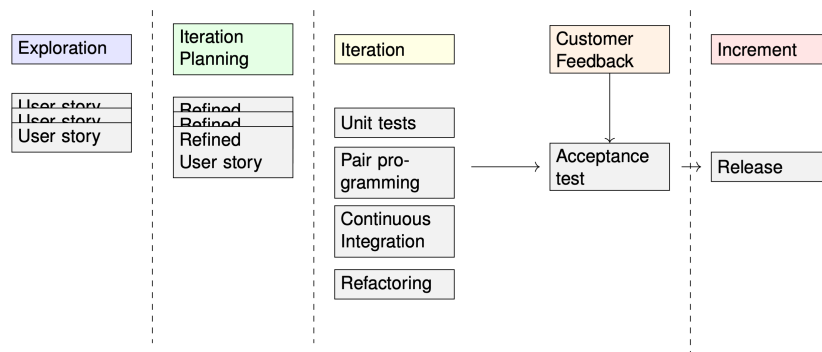
- **Customer availability**
- **Coding guidelines**
- **Test first:** Schreibe die unit tests am Anfang jeder Wiederholung
- **Pair Programming:** Driver/Pilot und Observer
- **Continuos Code Integration**
- **Collective Code Ownership:** Jeder kann alles verändern
- **Optimize last**
- **No Overtime**

Testing

- **Unit tests:** Vor jeder Iteration, ersetzt Dokumentation
- **Acceptance tests:** Das ganze System gegen Spezifikationen testen

Timeline overview

Tabelle 13: Timeline overview: XP



Requirements engineering

System modeling

Architectural design

Design patterns

Implementation

Software testing

Software evolution

Software project management

Software engineering in machine learning