

Introduction to Software Engineering

Felix Leitl

6. März 2024

Inhaltsverzeichnis

Software processes	5
Software specification	5
Requirements election and analysis	5
Requirements specification	5
Requirements validation	5
Software design and implementation	5
Architectural design	5
Database design	5
Interface design	5
Component selection and design	6
Software verification and validation	6
Component testing	6
System testing	6
Customer testing	6
Software evolution	6
Change anticipation	6
Change tolerance	6
Software development life circle	7
Life cycle phases	7
Software Process Models	7
Wasserfallmodell	8
Improved Waterfall model	9
V-Modell	10
Wiederverwendungsansatz	12
Agiles Modell	13
Change Management	13
Prototyp	13
Schrittweise Veröffentlichung	14

Agile software development	14
Agile manifesto	15
Rational Unified Process (RUP)	15
Vier Phasen	15
Disziplinen	16
Kanban	17
Practices	17
Extreme programming	18
Core values	18
Vier Aktivitäten	18
Planning	18
Design	19
Implementation	20
Testing	20
Timeline overview	20
Scrum	20
Principles	20
5 Scrum values	21
Timeline overview	21
Roles	21
Processes	21
Responsibilities	22
Product Backlog	22
Sprint Backlog	22
Dynamic Systems Development Method	22
Building Blocks	22
Principles	23
Crystal	23
Colros	23
Hardnesses	23
Crystal family	24
Principles	24
Required practices (Crystal Clear)	24
Roles (Crystal Clear)	24
Agility and large systems	25
Requirements engineering	25
Requirements	25
Definition	25
User requirements	26
System requirements	26
Functional requirements	26
Non-functional requirements	26
Key qualities of requirements	27
Requirements Engineering Process	28
Requirements Elicitation	29

Data gathering	29
Collaborative	29
Cognitive	29
Contextual	29
Creativity	29
Requirements Specification	29
Modeling techniques	29
Agile Requirements Specification	32
User stories	32
Acceptance criteria for User Stories	32
User Storie vs. Use Cases	32
Technical Stories	32
Requirements Validation	32
Validation Checks	32
Requirements validation techniques	32
Iterative testing	33
Requirements Evolution	33
Bug vs. Defect	33
Innovation vs. Tuning	33
Requirements management	33
Steps of requirements change management	33
System modeling	34
System Modeling	34
UML Diagrams	34
Interaction Models	34
Structural Models	35
Class diagrams	35
Behavioural Models	36
Data-driven model	36
Event-driven model	37
Context Models	37
Supporting models	38
Architectural design	38
Architectural design decisions	38
Non-functional requirements	38
Technical aspects	38
Organisational aspects	39
Architectural views	39
Principles	39
Architectural Patterns	40
Structuring patterns	40
Adaptable systems	42
Distributed systems	43
Interactive Systems	47

Design patterns	48
Implementation	48
Software testing	48
Software evolution	48
Software project management	48
Software engineering in machine learning	48

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
- Nuteranforderungen (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 dass ganze Produkt

Change tolerance

- Design berücksichtigt Veränderungen am System
- Normalerweise durch schrittweise Entwicklung
- Eine 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:

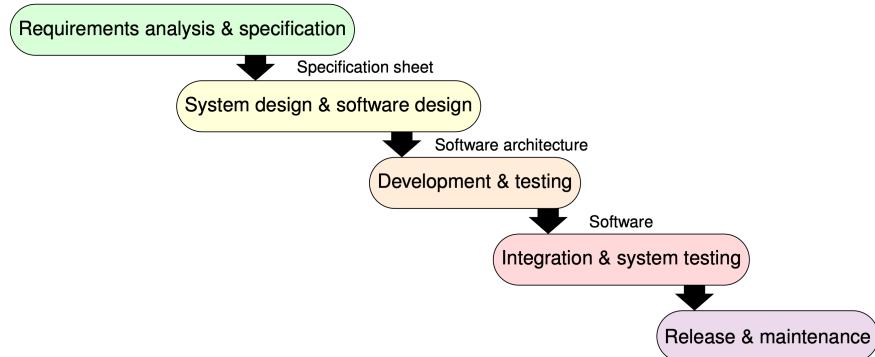
- Abläufe zu definieren
- die Ablaufordnung zu spezifizieren
- Phasen zu determinieren: Abläufe, Ziele, Rollen und Methoden

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



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

System & softwareedesign: 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

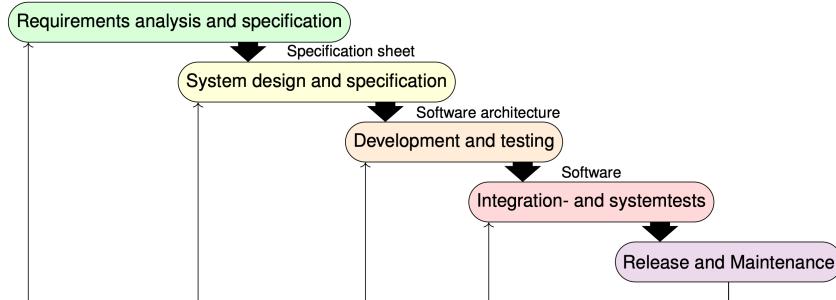
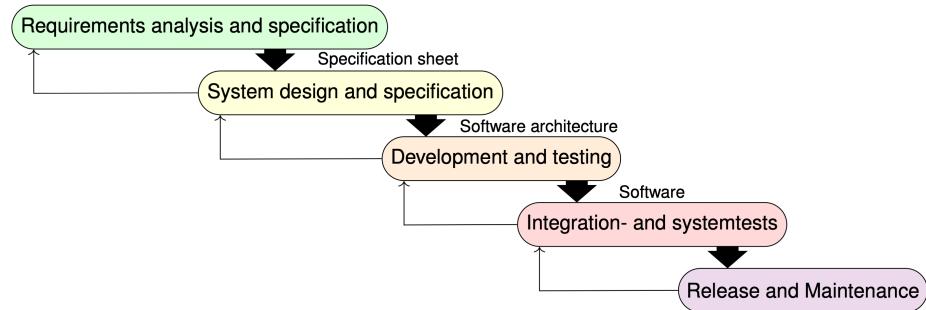


Tabelle 3: Incremental Waterfall model



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

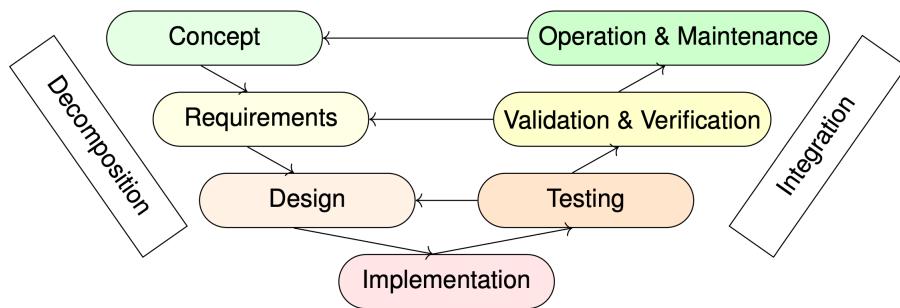
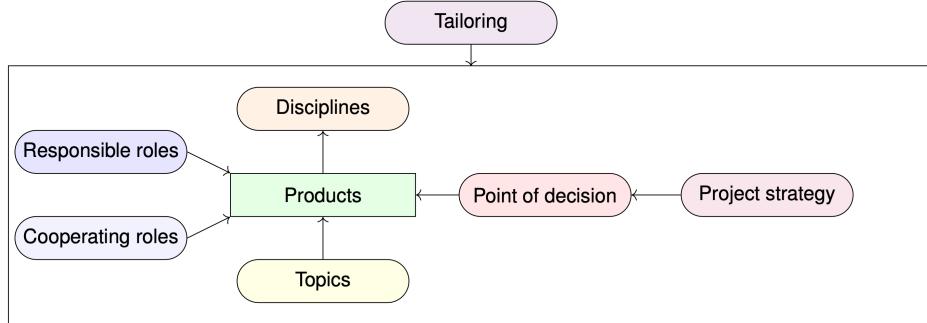
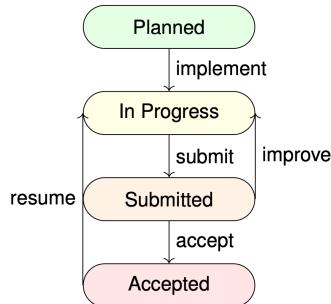


Tabelle 5: V-model XT



Jedes Produkt geht durch vordefinierte Zustände



Pros:

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

Cons:

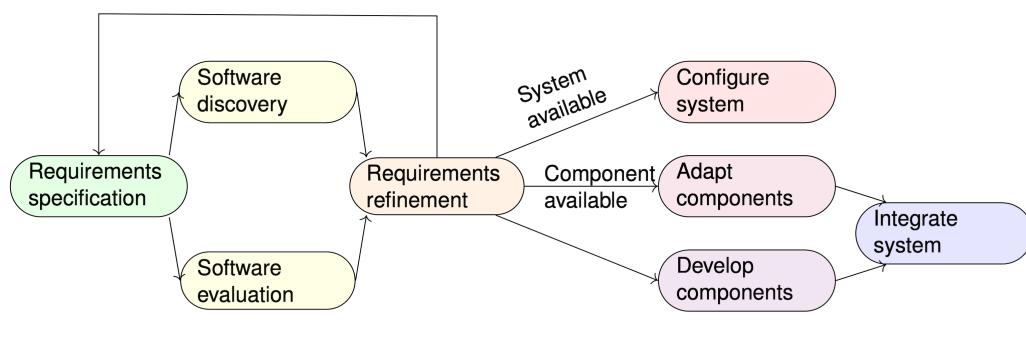
- Großer Overhead bei kleinen Projekten
- Testphase beginnt relativ spät
- Strikte Phasen
- Teilnehmer müssen geschult sein, um Modell zu folgen

Verwendung:

- Softwareentwicklung in Behörden
- Sicherheitsrelevante Projekte

Wiederverwendungsansatz

Tabelle 6: Reuse-oriented approach



Pros:

- Reduziert Kosten und Risiken
- Schnellere Verteilung

Cons:

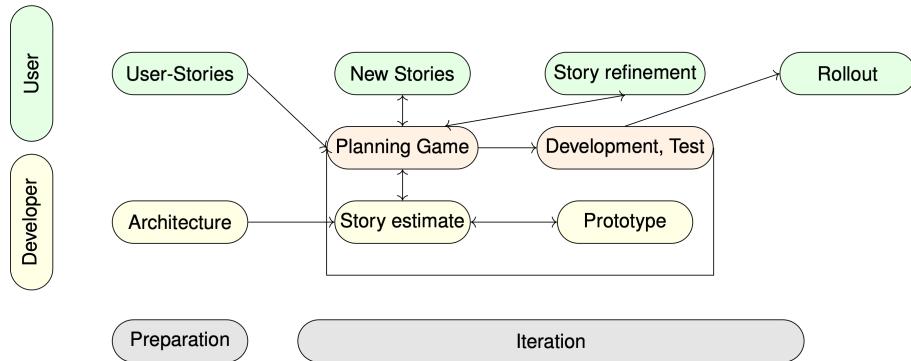
- Kompromisse in den Anforderungen
- System könnte echte Nutzerbedürfnisse nicht erfüllen
- Kein oder limitierte Kontrolle über Systemevolution

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

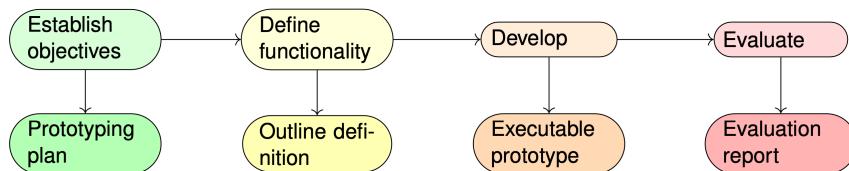
Verwendung:

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

Change Management

Prototyp

Tabelle 8: Prototyping



Pros:

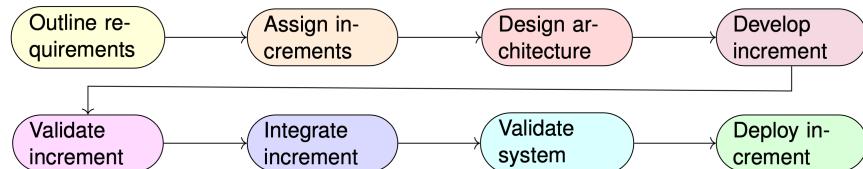
- Kunde gibt Priorität der Anwendung vor
- Software kann sofort verwendet werden
- Kunde erlangt Erfahrung mit dem System
- Kunde kann Anforderungen für spätere Schritte abgeben
- Veränderungen sind einfach umzusetzen
- Die wichtigsten Systemteile werden am häufigsten gestet

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

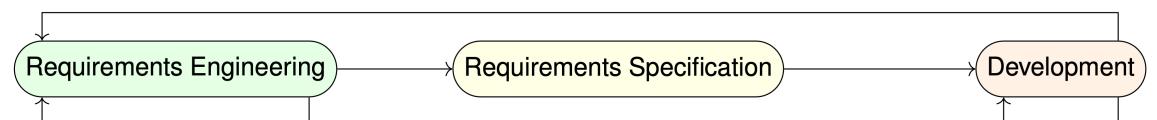
Tabelle 9: Incremental delivery



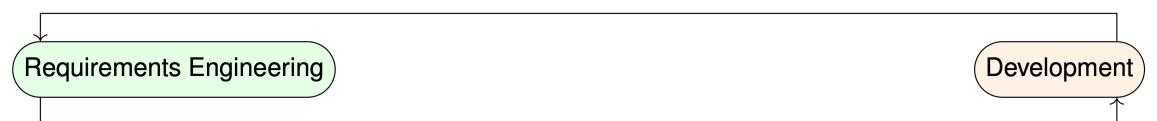
Agile software development

Tabelle 10: diff. between plan-driven and agile

Plan-driven:



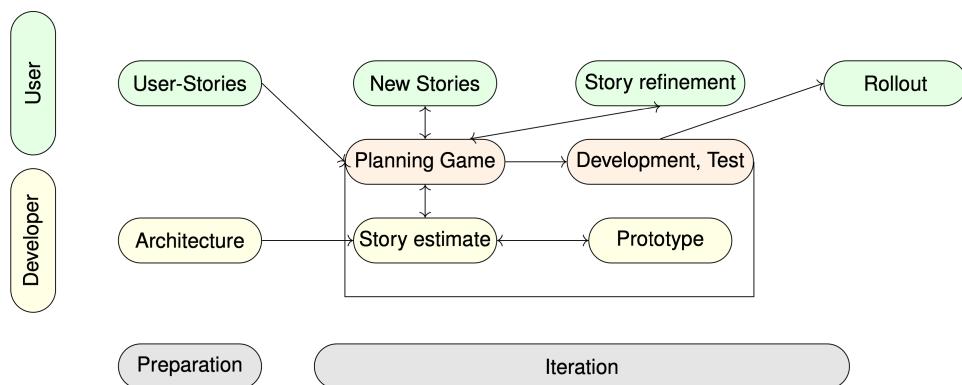
Agile:



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-

dellieren

Cons:

- Komplex
- Nicht flexibel
- Große Anzahl an Dokumenten

Kanban

Kanban verwendet eine visuelle, 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:	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Class Name Superclass</td><td style="padding: 2px;"></td></tr> <tr> <td style="padding: 2px;">Responsibility</td><td style="padding: 2px;"></td></tr> <tr> <td style="padding: 2px;">Collaboration</td><td style="padding: 2px;"></td></tr> </table>	Class Name Superclass		Responsibility		Collaboration		Back:	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Operations</td><td style="padding: 2px;"></td></tr> <tr> <td style="padding: 2px;">Attributes</td><td style="padding: 2px;"></td></tr> </table>	Operations		Attributes	
Class Name Superclass													
Responsibility													
Collaboration													
Operations													
Attributes													

Implementation

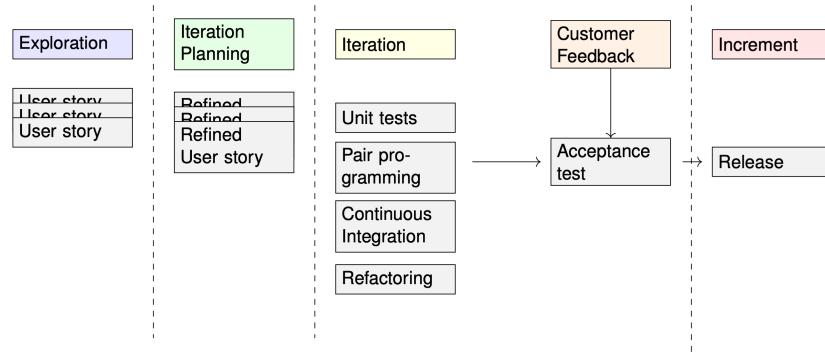
- Customer availability
- Coding guidelines
- Test first: Schreibe die unit tests am Anfang jeder Wiederholung
- Pair Programming: Driver/Pilot und Observer
- Continuos Code Integration
- Collective Code Ownership: Jeder kann alles verändern
- Optimize last
- No Overtime

Testing

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

Timeline overview

Tabelle 13: Timeline overview: XP



Scrum

Principles

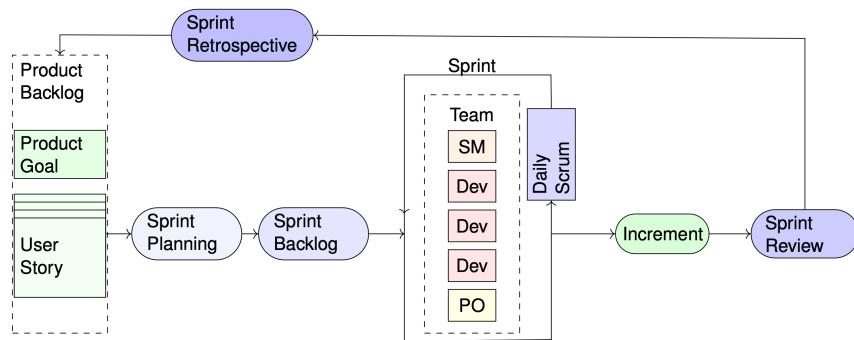
- Transparency
- Inspection
- Adaption: Prozess muss angepasst werden um gute Ergebnisse zu erzeugen

5 Scrum values

- Commitment
- Focus
- Openness
- Respect
- Courage

Timeline overview

Tabelle 14: Scrum



Roles

- Product Owner
- Stakeholder: Customers, user
- Development Team
- Scrum Master

Processes

- Sprint: Zeitfenster, um ein Increment zu entwickeln (max. 1 Monat)
- Sprint Planning: Anforderungen für nächsten Sprint planen
- Daily Scrum: 15 min, um Entwicklung des Tages zu planen
- Sprint Review: Increment validieren und Backlog anpassen
- Sprint Retrospektiv: Verbesserungen für den nächsten Sprint

Responsibilities

- Product Owner:
 - Produktziel formulieren und kommunizieren
 - Eindeutige Produkt-Backlogs erschaffen
 - Produkt-Backlog-Einträge ordnen
 - Sicherstellen, dass der Produkt-Backlog transparent und verständlich ist
- Development Team:
 - Sprint-Backlog erzeugen
 - Einhaltung der „Done“-Definition
 - Tägliche Anpassungen, um Sprintziel zu erreichen
- Scrum Master:
 - Techniken bereitstellen

Product Backlog

- eine geordnete Liste aller Produktanforderungen
- Die Einzige Quelle für Anfrageänderungen
- Niemals Vollständig
- Eine Mischung aus Features, Funktionalität, Veränderungen und Fehlern

Sprint Backlog

- Eine Prognose, der Funktionalitäten des nächsten Increments
- Eine Echtzeitvisualisierung, der Arbeit, des Entwicklungsteams
- Detaillierter Fortschritt des Sprints
- Wird ausschließlich vom Entwicklungsteam bearbeitet

Dynamic Systems Development Method

Building Blocks

- Philosophy
- Eight Principles
- Four P's: Process, People, Products, Practices
- Common sense and pragmatism

Principles

1. Focus on the business need
2. Deliver on time
3. Collaborate
4. Never compromise quality
5. Build incremental from firm foundations
6. Develop iteratively
7. Communicate constantly and clearly
8. Demonstrate control

Crystal

Crystal Clear ist nicht verwendbar, bei fail-safe und life-critical Systemen

Colros

Tabelle 15: Crystal colors

Crystal Clear	1 to 6 members
Crystal Yellow	Up to 20 members
Crystal Orange	Up to 40 members
Crystal Red	Up to 100 members
Crystal Maroon	Up to 200 members
Crystal Blue	Up to 500 members
Crystal Violet	Up to 1000 members

Hardnesses

C	Loss of comfort	Nutzbarkeit ist reduziert, aber Funktionalität noch gegeben
D	Loss of discretionary	Nichtkritische Finanzierungen sind verloren
E	Loss of essential money	Kritische Finanzierungen sind verloren
L	Loss of Life	Menschen sind gestorben

Crystal family

Tabelle 16: Crystal family

	L6	L20	L40	L100	L200	L500	L1000
Risk ↑	E6	E20	E40	E100	E200	E500	E1000
	D6	D20	D40	D100	D200	D500	D1000
	C6	C20	C40	C100	C200	C500	C1000
	1 - 6	up to 20	up to 40	up to 100	up to 200	up to 500	up to 1000
	→ Team size						

Principles

- Regular delivery
- Reflective improvement: Vorherige Entwicklungsphase reflektieren
- Osmotic and condescended communication
 - Osmotic: Kommunikation in kleinen Teams
 - Condensed: Kommunikation in großen Teams durch Bildung kleiner
- Personal security: Ehrlich ohne Angst, Fehler eingestehen
- Choose priorities: Management wählt und kommuniziert
- Easy communication with user
- Good engineering environment: Autotests, regelmäßige Integration

Required practices (Crystal Clear)

- Osmotic communication, ideally the team works in one room
- Increment cycles of less than four months
- User is involved in Project
- Project assignment is defined with high-level system design
- Responsibilities need to be clearly defined

Roles (Crystal Clear)

Most important roles:	Less important roles
• Customer	• Coordinator
• Experienced user	• Field expert
• Senior architect	• Tester
• Designer / Developer	• Author

Agility and large systems

Tabelle 17: Large scale Problems

Principle	Practice
Customer involvement	Dependent on the customer, who often can not be involved full time
Embrace change	Each stakeholder has different, often conflicting priorities
Incremental delivery	Business and marketing side plans long-term
Maintain simplicity	Pressure due to deadlines
People, not process	Members may not have fitting personalities

Um agile Modelle bei Großprojekten anzuwenden werden diese meist in plan-driven Ansätze integriert.

??

Requirements engineering

Requirements

Definition

1. Ein nötiger Zustand oder eine nötige Fähigkeit, eines Nutzers, um Probleme zu lösen oder ein Ziel zu erreichen
2. Ein nötiger Zustand oder eine nötige Fähigkeit, die von einem System erreicht, bzw. erlangt werden müssen, um einen Vertrag, Standart, eine Spezifikation oder Formalie zu erfüllen
3. Eine dokumentierte Repräsentation des Zustands oder der Fähigkeit wie (1) oder (2)

User requirements

- Aussagen in natürlicher Sprache und Diagramme von Anwendungen, die das System voraussichtlich bekommen soll und die Vorlagen nach welchen das System sich verhalten zu hat
- Beschreibt oft das externe Verhalten des Systems, wie input und output
- Kann auch convenience features, welche das handling der Software erleichtern, beschreiben

Informell: Was soll passieren

System requirements

- Eine detaillierterer Beschreibung der Systemfunktionalität und der Verwendungsbeschränkung
- Nutzeranforderungen können zu mehreren Systemanforderungen erweitert werden, indem mehr Informationen über den Service und die Funktionalität des Systems bereitgestellt werden

Informell: Wie soll es passieren

Functional requirements

- Funktionalität des Systems, die implementiert werden sollen
- Services die das System bereitstellen sollte
- Wie es in bestimmten Situationen reagieren sollte
- In manchen Fällen auch, was es nicht tun sollte

Non-functional requirements

- Alle Anforderungen, die nicht direkt als Funktion implementiert werden
- Einschränkungen der Systemservices, wie Zeitbeschränkungen
- Einschränkungen, die die Entwicklung betreffen
- Einschränkungen, die von Standards vorgeschrieben werden

Classification:

- product
 - Usability requirements
 - Security requirements
 - Dependability requirements

- Efficiency requirements
- Organisational requirements
 - Operational requirements
 - Environmental requirements
 - Development requirements
- External requirements
 - Regulatory requirements
 - Ethical requirements
 - Legislative requirements

Key qualities of requirements

- Measurability
- Completeness
- Correctness
- Consistency
- Unambiguity
- Pertinence
- Feasibility
- Traceability
- Comprehensibility
- Modifiability

Requirements Engineering Process

Tabelle 18: Main activities

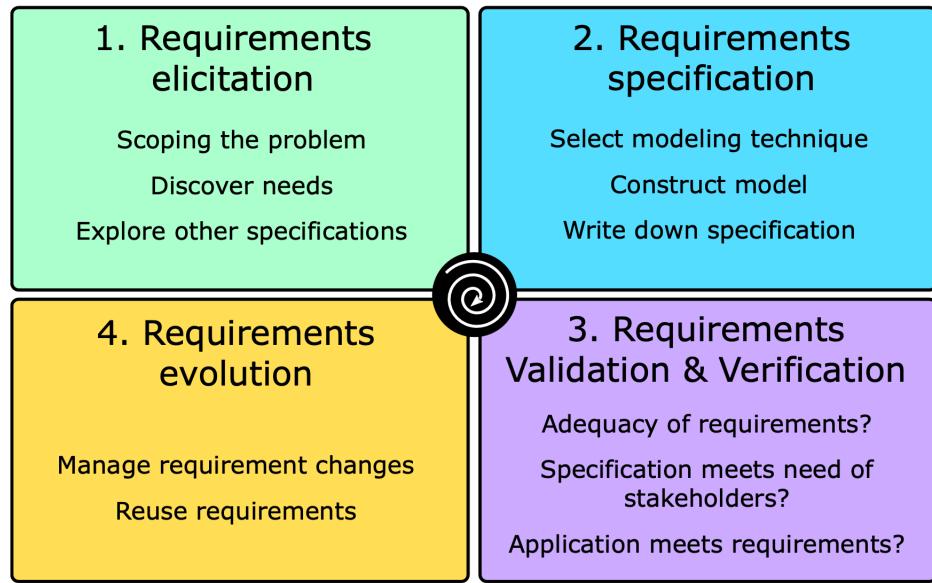
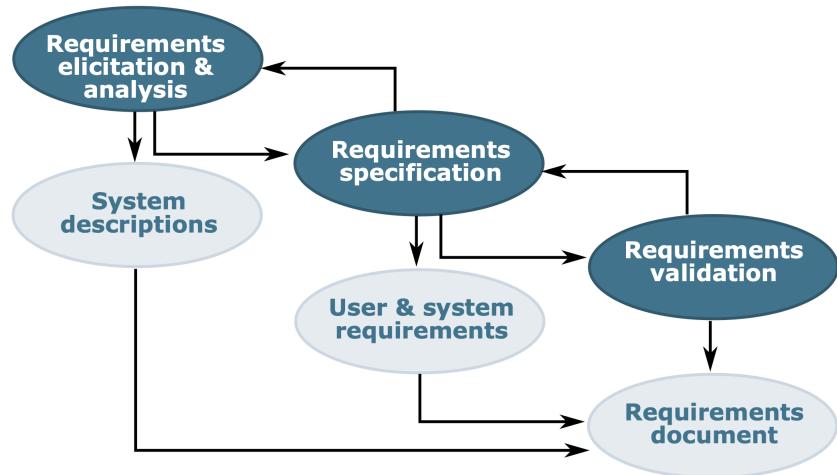


Tabelle 19: Main Activities & Documents



Requirements Elicitation

Problem: Kommunikation zwischen Kunden und Entwicklern ist kompliziert, durch die nicht überschneidenden Expertisen

Data gathering

- Background study
- Interviews
- Questionnaire

Collaborative

- Brainstorming
- Joint application development (JAD) workshops
- Rapid application development (RAD) workshops

Cognitive

- Repertory grids
- Card sorting

Contextual

- Observation
- Protocol analysis

Creativity

- Creativity workshops
- ContraVision

Requirements Specification

Modeling techniques

- Natural language specification
 - Easy Approach to Requirements Syntax (EARRS)
(while, when, where, if then: shall)
 - MoSCoW (Must, Should, Could & Won't)
- Structural modeling (what)

- Problem Frames
- Class diagrams
- Entity-Relationship (ER) diagrams
- Behavioural modeling (how)
 - Use cases: semi-formal
 - (Finite) state machines: formal
 - Petri nets: formal
- Goal modeling (why & who)
 - KOAS
 - i*

Tabelle 20: Problem diagram

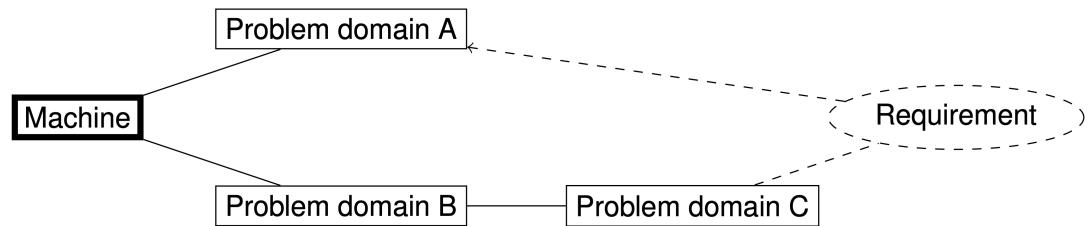
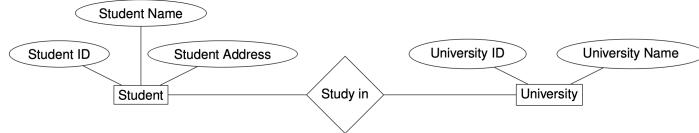


Tabelle 21: ER-diagram



Notation:

- **Entity**: rectangle; is an object or component of data
- **Attribute**: ellipse; describes property of entity
- **Relationship**: diamond; describes relationship among entities; can be 1-to-1, 1-to-many, or many-to-many relationships ("cardinality" → more about that in the next lecture unit "System modeling")

Attributes are categorized into:

- **Key attribute**: ellipse with text underlined; uniquely identifies entity from entity set, e.g. student ID
- **Composite attribute**: is a combination of other attributes, e.g. address
- **Multivalued attribute**: double ellipse; can hold multiple values, e.g. a person can have more than one phone number
- **Derived attribute**: dashed ellipse; value is dynamic and is derived from other attribute, e.g. age derived from date of birth

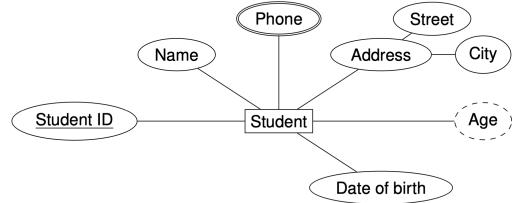
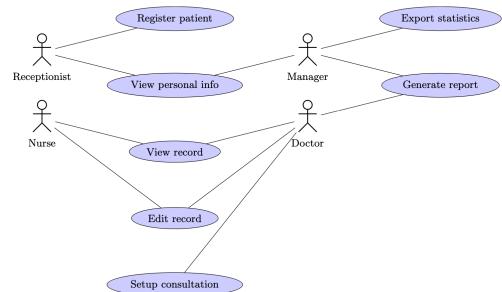


Tabelle 22: Use case diagram

Use cases are modeled using a graphical model and structured text (UML Use Case Diagram):

- Actors (e.g. humans or other systems): Stick figures
- Class of interaction: Named ellipses
- Association: Line links between actors and interactions (optionally with arrowheads to show the initiator of the action)



Agile Requirements Specification

User stories

As <user role>, I want <functionality/system behaviour>, so that <technical value for the user/customer or economic benefit>

Acceptance criteria for User Stories

- SMART: Specific, Measurable, Achievable, Relevant, Time-boxed
- Given-When-Then scheme

User Stories vs. Use Cases

Allgemeiner Verwendungszweck von User Stories und Use Cases:

- Systemfunktionalität durch Nutzerperspektive modellieren
- Überblick über das System

Unterschiede:

- Ein Use Case deckt mehrere User Stories oder ein Epic ab

Technical Stories

- Decken technische oder nichtfunktionale Aspekte ab
- Optional, könnte auch in der Definition von Done definiert werden
- Haben eine niedrigere Priorität als User Stories, da der Kunde nicht direkt dafür zahlt

Requirements Validation

Validation Checks

- Validity checks: Erfüllen die Anforderungen die needs der user
- Consistency checks: Keine Konflikte zwischen Anforderungen
- Completeness checks
- Realism checks
- Verifiability

Requirements validation techniques

- Requirements reviews
- Prototyping
- Test-case generation

Iterative testing

Wird verwendet um bei stetig wachsenden Projekten Fehler früh zu erkennen und zu beheben

Requirements Evolution

Bug vs. Defect

Ein Bug erfüllt die dokumentierten Anforderungen nicht und erfordert Korrektur. Ein Defekt erfüllt die dokumentierten Anforderungen, allerdings nicht die tatsächlich benötigten und erfordert Korrektur der Anforderungen und des Systems

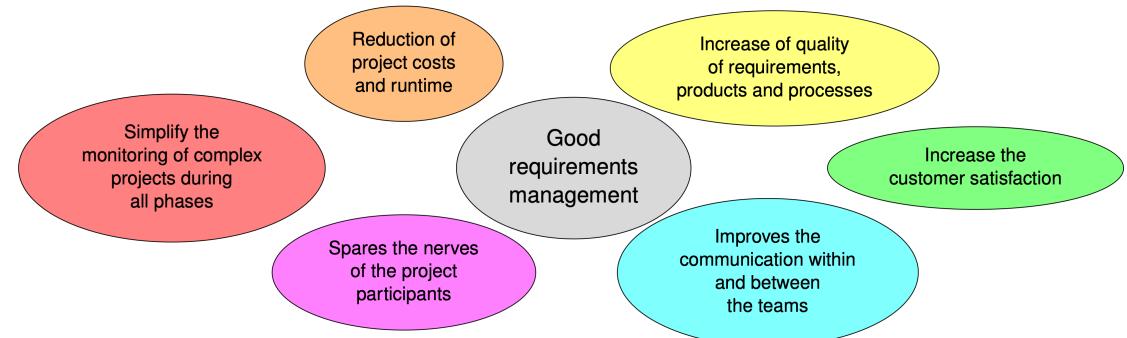
Innovation vs. Tuning

Bei Innovation werden neue Systemeigenschaften eingeführt, beim Tuning wird nur unter der Oberfläche modifiziert

Requirements management

Anforderungsmanagement erfordert Anforderungsanalysen und weiteren Nutzen der Anforderungen. Bei agilen Modellen einfacher umzusetzen, da nicht so formell und inkrementell

Tabelle 23: Requirements management



Steps of requirements change management

1. Anforderungsproblem identifizieren oder Änderungsvorschlag
2. Problemanalyse: Überprüfen, ob Änderungsvorschlag valide ist
3. Analyse und Kosten anpassen
4. Implementation anpassen

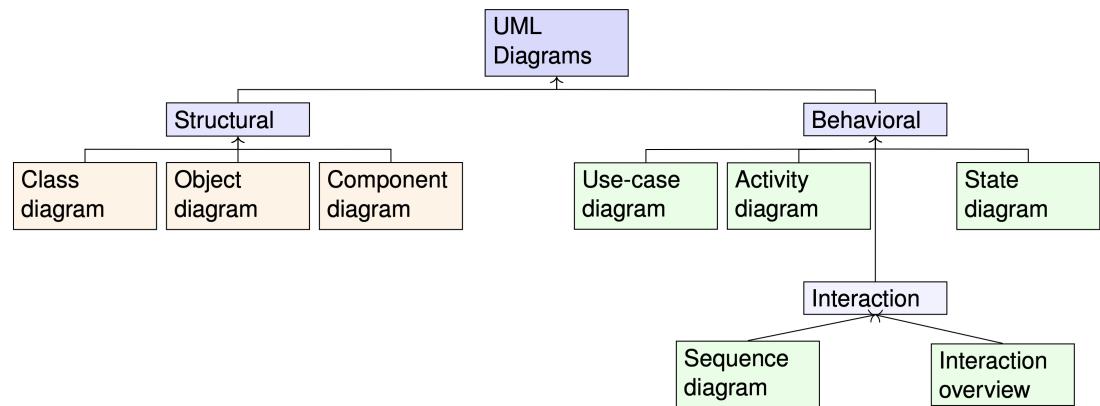
System modeling

System Modeling

Werden verwendet, um verschiedene Ansichten des Systems zu präsentieren. Es ist eine abstrakte Repräsentation und Simplifizierung des echten Systems. Details werden je nach Perspektive ausgelassen. Z.B. Kontext, Struktur Verhalten oder Interaktion.

UML Diagrams

Tabelle 24: UML Diagrams



Interaction Models

Können für Nutzerinteraktionen, System-zu-System-Interaktionen oder Komponenteninteraktionen verwendet werden.

Interaktionen können mit **Use case** oder **Sequenz Diagrammen** gemodelt werden

Tabelle 25: Use Case

Use cases are modeled using a graphical model and structured text (UML Use Case Diagram):

- Actors (e.g. humans or other systems): Stick figures
- Class of interaction: Named ellipses
- Association: Line links between actors and interactions (optionally with arrowheads to show the initiator of the action)

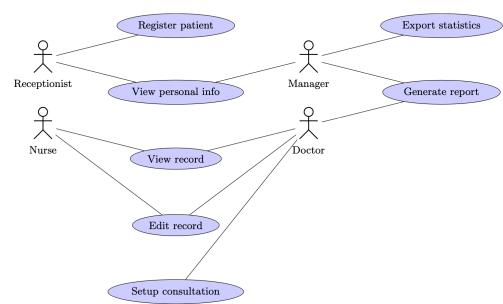
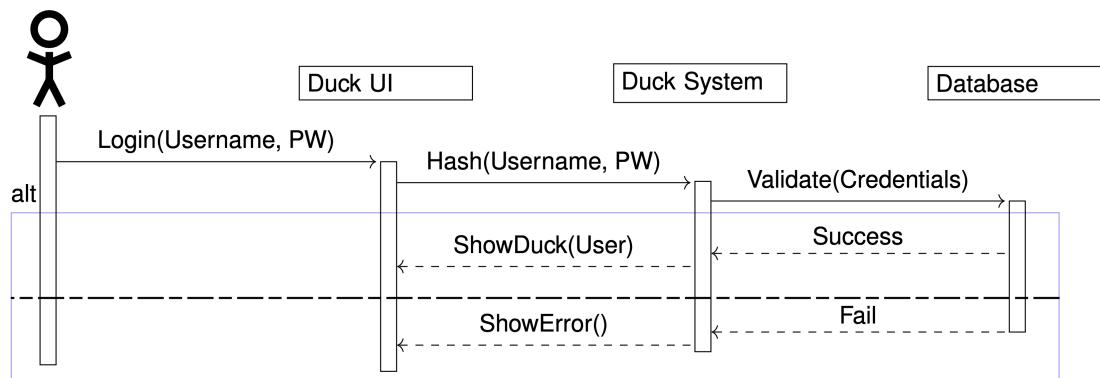


Tabelle 26: Sequence



Structural Models

Werden während der Designphase erzeugt und zeigen alle Komponenten und ihre Relationen

Class diagrams

Classes

1. Class name: Customer
2. Atributes: +id: int
3. Methode name: +inquiry(): void

Connection

- Association: Relation zwischen zwei Klassen (-)

- Directed Association: Richtung mit einem Pfeil (\rightarrow)
- Dependency: Veränderung könnte Veränderung der anderen Klasse verursachen ($-- \rightarrow$)
- One to One: (1 – 1)
- One to X: (1 – 1..4)
- One to many: (1 – 1...*)
- Inheritance: Nicht ausgefüllter Pfeil zu Überklasse
- Aggregation: (Whole $\diamond-$ Part)
- Composition: (Whole (Filled-) $\diamond-$ Part)

Behavioural Models

Data-driven model

Tabelle 27: Activity model

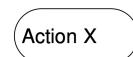
Start/ End

- Start or end point of the activity



Action

- Action, Activity or process step



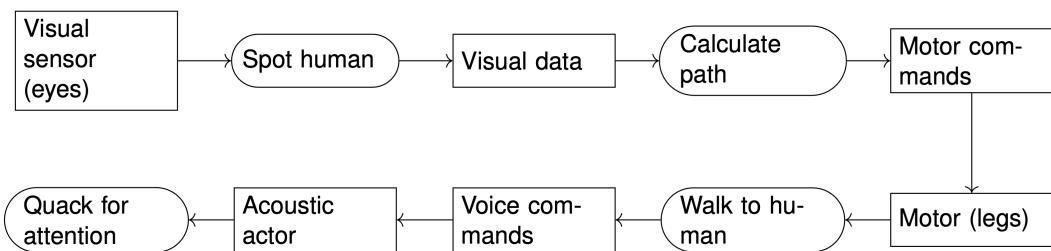
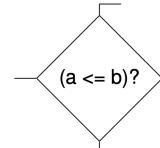
Control flow

- Shows the direction of the process



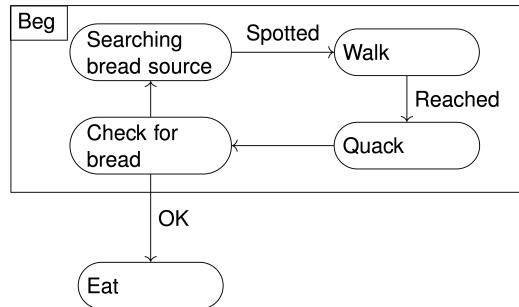
Case

- Arrows emerging from diamond, each is a different case
- Case needs to be annotated at arrow



Event-driven model

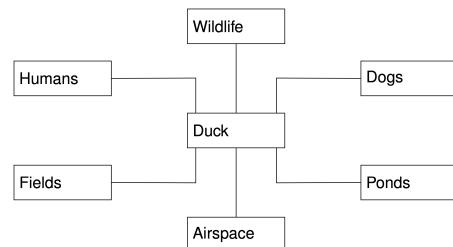
Tabelle 28: State diagram



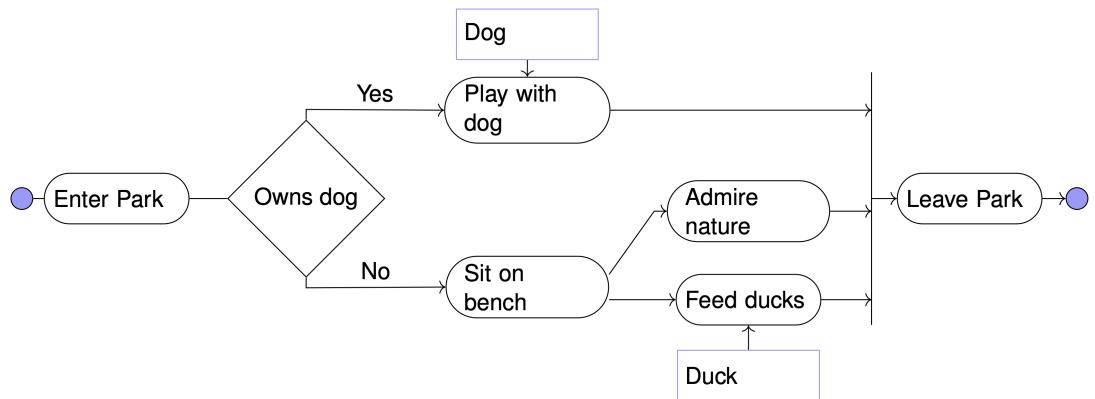
Context Models

Kontextmodelle modellieren die Umwelt und das System, jedoch nicht die Art der Relationen, räumliche Relationen, geteilte Daten und wie das System sich verbindet.

Tabelle 29: Context diagram



Supporting models



Architectural design

Besteht aus den statischen Komponenten des Systems und der dynamischen Beschreibung ihrer Interaktionen, sowie der Strategie der Architektur

Architectural design decisions

Non-functional requirements

- Performance
- Security
- Safety
- Availability
- Maintainability
- Simplicity
- Scalability
- Testability
- Expandability
- Reliability

Technical aspects

- System structure
- Interfaces

- Dynamic behaviour

- | | |
|---|--|
| <ul style="list-style-type: none"> • Patterns • Used Products • Used Tools • Testing strategy | Organisational aspects <ul style="list-style-type: none"> • Use of human resources • Priorization of use case • Cost |
|---|--|

Architectural views

- Logical View
 - Hauptabstraktion sind Objekte und Klasse
 - Verwandt mit Systemanforderungen
- Development View
 - Komponenten die implementiert werden müssen
 - Wird von Softwaremanagern und Entwicklern verwendet
- Process View
 - Interagierende Prozesse zur Laufzeit
 - Wird genutzt, um Performance und Verfügbarkeit zu bewerten
- Physical View
 - Verteilung der Software auf die Prozessoren
 - Wird verwendet um die Verteilung zu planen

Diese Ansichten sollten verwendet werden um die finale Architektur zu dokumentieren

Principles

- Divide and conquer
- Design to test: Designed, sodass es einfach zu testen ist
- KISS: Keep it simple stupid
- Yagni: You aren't gonna need it
- DRY: Don't repeat yourself
- Single level of abstraction principle: Alle Aussagen in einer Methode sollten das selbe Level an Abstraktion haben
- Loose coupling and strong cohesion
- Abstraction

- Information hiding
- Separation of concerns
- Dependency Inversion Principle
- Interface Segregation Principle
- Design by contract
- Liskovsch's Substitution Principle: Eine Referenz zu einer Elternklasse sollte immer durch eine Referenz zu einer Kinderklasse ersetzt werden können
- Principle of Least Astonishment
- Open-Closed Principle: Ein Modul sollte offen für Erweiterungen und geschlossen für Veränderungen sein
- Develop against Interfaces, not Implementations
- Inversion of control

Architectural Patterns

Structuring patterns

Layers:

- Jede Schicht wird mit einer Funktionalität assoziiert
- Eine Schicht kann auf Services der unteren Schicht durch ein Interface zugreifen
- Die niedrigste Schicht stellt die Kernfunktionalität
- Jede Schicht ist nur von der direkt untern abhängig

Pro:

- Schichten sind einfacher zu verstehen, als das ganze System
- Schichten können wiederverwendet werden
- Schichten sind stabil und können standardisiert werden
- Abhängigkeiten können minimiert werden
- Schichten können parallel entwickelt werden
- Schichten können bottom-up getestet und implementiert werden
- Schichten können, bei gleichem Interface, ersetzt werden

Cons:

- Die richtigen Schichten klar zu treffen ist schwer
- Striktes Schichten kann limitieren
- Zugriff über Schichten kostet mehr Zeit
- Veränderungen können sich auf mehrere Schichten auswirken

Verwendung:

- Hardware, OS und Anwendungssoftware
- Embedded Systems
- Entwicklung auf Basis eines existierenden Systems
- Viele Teams arbeiten an unterschiedlichen Komponenten
- Multilevel security

Pipes and Filter

- Pipes
 - Können Daten zwischenspeichern
 - Asynchrones entkuppeln
 - Liefert Daten erst wenn benötigt
- Filters
 - Ein Filter pro Verarbeitungsschritt
 - Nimm Daten von vorherigen Filtern
 - Transformiert die Daten zum Output
 - Kann Daten entfernen, hinzufügen oder modifizieren
- Pipeline
 1. Data source
 2. Filter and Pipes
 3. Data sink
- Active Filter
 - Nimmt sich aktiv die Daten
 - Sendet die Daten an das nächste Element
 - Unabhängiger paralleler Prozess
 - Sind mit einer Pipe gekuppelt, um Verarbeitungsgeschwindigkeiten auszugleichen

- Passive Filter
 - Kann direkt gekuppelt werden
 - Push-principle
 - * Empfängt passiv Daten
 - * Verhält sich wie eine void-Funktion
 - Pull-principle
 - * Daten werden vom nächsten Element geholt
 - * Verhält sich wie eine Funktion

Pros:

- Flexibel
- Wiederverwendbar
- Erlaubt schnelles Prototyping
- Entkuppelt Komponenten
- Zwischenergebnisse zu speichern ist möglich aber nicht nötig
- Bis zu einem Grad parallelisierbar
- Evolution durch hinzufügen von Transformationen
- Workflow entspricht Businessprozessen

Cons:

- Globale Daten sind nicht einfach zu verwenden
- Passive Filteraufrufe sind hard coded
- Bugs sind schwer zu finden
- Flaschenhals bei langsamsten Filter
- Overhead von Datenkonversation
- Erhaltung kann dich durch Komponenten ziehen

Repositorys Eine Gruppe an Komponenten die sich die selben Daten teilen, welche zentral in einem Repository gemanaged werden

Pros:

- Unabhängige Komponenten
- Datenänderungen für alle verfügbar
- Konsistentes Datenmanagement

Cons:

- Single point of failure
- Flaschenhals in der Kommunikation
- Verbreitung kann schwierig sein

Adaptable systems

Plug-In orientiert sich am Open-Closed-Prinzip **Plug – In**

- Softwarekomponente, die zusätzliche Funktionalität bietet
- Unabhängig von App

- Komplettes Entkoppeln von spezial Applications
- Normalerweise nicht eigenständig ausführbar
- Kann kaskadiert werden

Plug – In manager

- Managed Anzahl und Art der Nutzung, der Plug-Ins
- Sucht nach passenden Plug-ins
- Instaziiert Plug-Ins zur Laufzeit, falls nötig

Pros:

- Separation of concerns
- Robust
- Erweiterung ohne Kenntnis des Codes
- Schlanke System
- Einfach zu erhalten
- Einfach zu verbreitende Implementation
- Versionsmanagement ist möglich
- Unabhängiges testen jedes Plug-Ins

Cons:

- Initialer Implementationsaufwand ist höher
- Overhead während der Ausführung
- Design allgemeiner Interfaces kann schwer sein

Distributed systems

Client – Server

- Client
 - Calls on Server
 - Thin-client
 - * Nur eine Schicht bei Client
 - * Normalerweise Input oder Visualisierung
 - * Server verarbeitet und speichert die Daten
 - Fat-client
 - * Datenverarbeitung auf Client-Seite
 - * Server wird nur für Datenspeicherung verwendet
- Server
 - Bietet Service an
 - Reagiert auf Client-Anfragen

Pros:

- Verteiltes System
- Ein Server dient mehreren Clients
- Thin client
 - Client-Management ist einfach
 - Minimale Anforderungen für den Client

Cons:

- Single point of failure von jedem Service
- Performance hängt von Netzwerk ab
- Management kann problematisch sein
- Thin client
 - Verarbeitungsbelastung beim Server
 - Könnte höhere Netzwerk- und Serverkapazitäten erfordern

Um ein messer skalierbares System zu erhalten kann man multi-tear-server verwenden, wobei Die Server inklusive Client unterschiedliche Aufgaben erledigen (z.B. Three-Tear: Client (I/O), Processing Server, Data Server)

Tabelle 30: Three-tear-client-server model

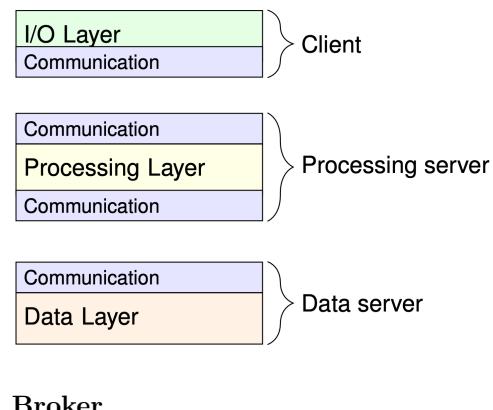
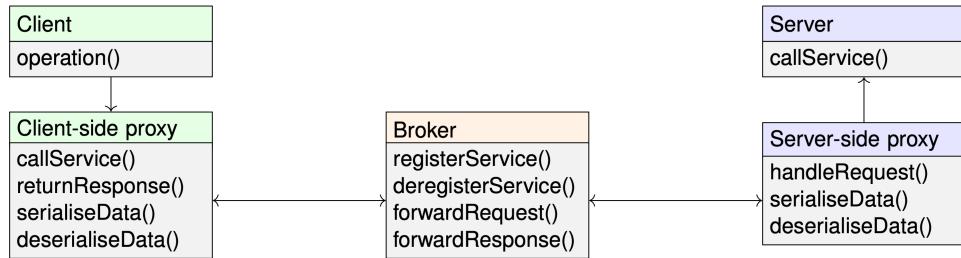


Tabelle 31: Broker



Pros:

- Separation of concerns
- Stabilität
- Räumliche Unabhängigkeit
- Plattform-Unabhängigkeit

Cons:

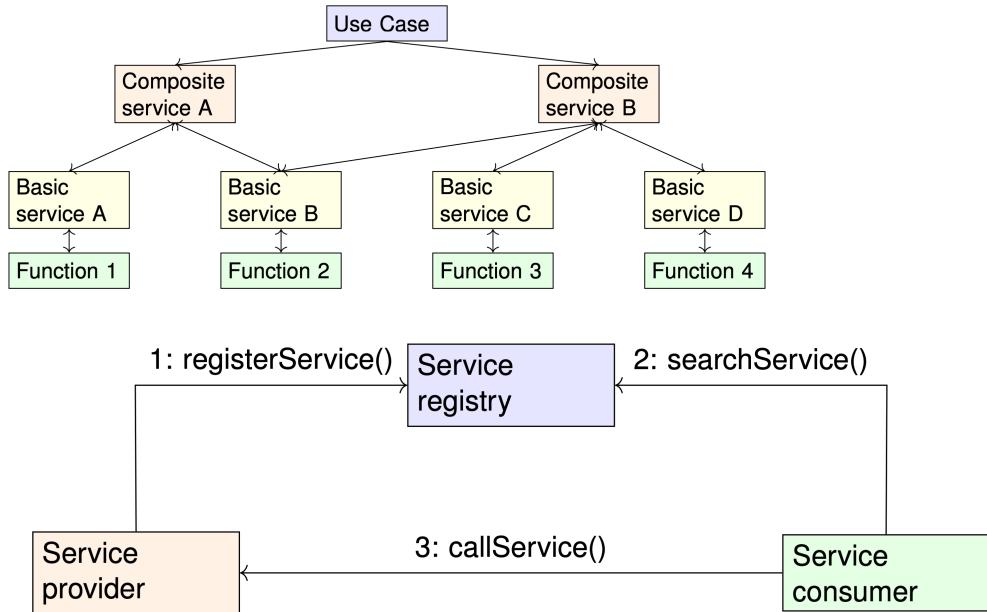
- Fehlertoleranz ist nötig
- Indirekte Kommunikation führt zu Performanceverlust
- Potenzieller Flaschenhals beim Broker
- Proxies sind von Broker abhängig

Verwendung

- Client und Server müssen entkoppelt sein
- Komponenten müssen Services durch Servicenamen erreichen
- Komponenten ändern sich während der Laufzeit
- Implementationsdetails von Client und Server müssen verborgen bleiben

Service – Oriented Architecture (SOA)

Tabelle 32: SOA



Eigenschaften

- Verteilt
- In sich geschlossen
- Zustandslos
- Leicht gekoppelt
- Austauschbar
- Ortstransparent
- Platformunabhängig
- Durch ein Interface erreichbar
- Registriert in z.B. einem service directory

Pros:

- Use cases sind an Komponenten gebunden
- Überblick über erforderliche Services und Interfaces
- Komplexität ist für verteilte Systeme reduziert
- Wiederverwendbarkeit
- Hält veränderten Anforderungen

stand

Cons:

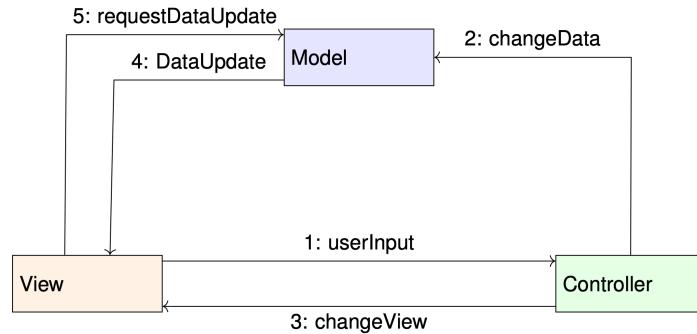
- Feingranulare Services führen zu komplexen Strukturen
- Overhead für Kommunikation
- Hohe Anforderungen für das Netzwerk
- Nur effektiv bei klar definierten und dokumentierten use cases

Interactive Systems

Model – View – Controller (MVC)

- Model
 - Speichert Systemdaten
 - Managt Interaktionen mit Systemdaten
- View
 - Definiert, wie Daten dargestellt werden
 - Managt Systemoutput
- Controller
 - Managt Input
 - Behandelt Nutzerinteraktionen

Tabelle 33: Model-View-Controller



Pros:

- Daten verändern sich unabhängig von der Repräsentation
- Unterstützt mehrere Ansichten auf die gleichen Daten
- Änderungen in einer Repräsentation werden in allen gesehen
- Model kann unabhängig der UI entwickelt und getestet werden
- UI kann einfach getauscht werden

Cons:

- Relativ großer Overhead für einfache Interaktionen
- Abhängigkeiten zwischen View und Controller
- Angestiegene Komplexität

Design patterns

Implementation

Software testing

Software evolution

Software project management

Software engineering in machine learning