SEPM V0

1 Einführung in SE: SE -> Vorgehensweise zur systematischen Erstellung von Software

Produkttypen: Kommerzielle Software
Eingebettete Systeme
Wissenschaftliche Software

-> Reifegrade der Software Entwicklung: CMMI

Software-Typen: Schwerpunkte zwischen: UI, Geschäftslogik, OS

Komplexitätstreiber: Größe
Team
Verwendete Technologien
Komplexität

Entscheidung: Kauf, Modifizierung, Neuerstellung
↳ Faktoren: Entsprechende Anforderungen, Änderbarkeit, Preis

-> Warum Projekte scheitern
-> Faktoren für erfolgreiches Projekt

Prozessmodelle: V-Modell
Inkrementelles Modell -> Spiralmodell
Iteratives Modell
Extreme Programming

Umfeld SE-Prozess: Projekt
Personen
Markt
Technologie

Planung, Kontrolle, Steuerung

2 Einführung in PM

PM: PM → einmaliges Vorhaben mit definiertem Anfang / Ende und mehreren Personen

Abgrenzbarkeit: Aufgabe / Erwartetes Ergebnis
Ressourcen
Zeitrahmen

Risiken:
- Analyse: Spezifisch, Generell; Wahrscheinlichkeit, Schaden
- Planung: Vorbeugemaßnahmen, Aktivitätsmaßnahmen

Teufelsquadrat: Qualität
Quantität
Entwicklungskosten
Kosten
→ Zeit nicht durch Personal kompensierbar

Elemente: Projektstrukturplan (was ist zu tun)
Arbeitsstrukturen (wer ist wofür zuständig)
Informationswesen (wer informiert wen, wann, wie)

Projektstart: Von Idee bis zu Kick Off
- Erarbeitung Projektidee
- Erstellung Ressourcenanforderung
- Vorstudien
↳ Projektumfeldanalyse

→ Projektauftragsklärung in Kick Off-Besprechung

Projektauftrag:
- Name
- Beschreibung
- Rollen und Verantwortl.
- Nutz-Ziele, Abgrenzung
- Komponentenübergr.
- Technologiewahl
- Qualität, Anforderungen
- Zeit-, Kostenplan
- Umfeld, Risikoanalyse

Rollen: Auftraggeber + Auftragnehmer
Projektleiter
Entwicklerteam: Architekt, Tester, Dokumentationsverantwortlicher, ...

Projektauftrag: schriftliche Zielvereinbarung zw. Auftraggeber und Nehmer um Projekt zu bestimmten Bedingungen durchzuführen
→ Ausarbeitung eines Projektvorschlags durch PM

Projektumsetzung: Klassisch / Sequenziell: Wasserfall, V-Modell
Iterativ: RUP
Agil: SCRUM, Extreme Programming

Techniken Projektsteuerung: Meilensteine
Balkendiagramme (Gantt)
Meilensteintrendanalyse
Burn Down Chart

PM: Gesamtheit von Methoden und Verhaltensweisen zur effizienten Steuerung der Entwicklung von besonderen Aufgabenstellungen

Vorgehensweise Auftrag: PSP erarbeiten
Arbeitspakete definieren
Aufwandsabschätzung
Terminplanung
Ressourcenplanung
Kostenplanung
Optimierung des Gesamtprojekts

3 SE Phasen

Vorgehensweise: - Systematische Vorgehensweise durch Software-Prozesse
→ wenn soll welches Produkt (Fertigstellungsgrad) erstmalig neu
- Konstruktive Methoden zur Herstellung von Softwareprodukten
→ Methoden für Spezifikation, Testfälle, Source-Code, ...
- Analytische Methoden zur Überprüfung der Produktqualität
→ Reviews, Inspektionen, Tests

Software-Lebenszyklus: Basiskonzept für SE Prozesse und Vorgehensmodelle

- Anforderungen + Spezifikation } Software Spezifikation
- Planung
- Entwurf und Design } Design und Implementierung
- Implementierung + Integration
- Betrieb + Wartung } Software Validierung
- Rückzug } Software Evolution

→ Requirements: Wünsche des Kunden (user / customer view)
→ Testen
→ Spezifikation: System aus technischer Sicht (engineering view)
→ Planung: Projektplanung bzgl. Zeit, Dauer, Kosten (project management)
→ Entwurf / Design: technische Lösung der Anforderungen
→ Komponenten, Plattformen, Datenstrukturen

Anforderungen und Spezifikation: gemeinsames Verständnis
Berücksichtigung unterschiedlicher Stakeholder
Beschreiben oder Grafiken darstellen
↳ Use Case der UML Familie
Testen und nachvollziehen

→ Funktionale Anforderungen
→ Nicht-funktionale Anforderungen: Performance, Usability, Sicherheit, Wartbarkeit, Erweiterbarkeit...
→ Designbedingungen: Schnittstellen, Entwicklungsumgebung, Verteilte Entwicklung
→ Prozessbedingungen: Ressourcen, Dokumentation

→ Priorisierung von Anforderungen: must-be
expected
nice-to-have

Entwurf und Design: definiert Architektur, Komponenten, Interfaces...

→ Software Architectural Design: Top-Level, Struktur
→ Software Detailed Design: detaillierte Beschreibung jeder Komponente
→ Grundlage der Implementierung, Sprachen der Techniker

Logical View → Implementation View
↓ Use Case View (Szenarios) ↓
Process View → Deployment View

- Logical View: Funktionale Anforderungen; Fokus End User
  -> Klassendiagramm, State-Machine, Package-Diagram
- Process View: Nicht-funktionale Anforderungen: Fokus Systemintegration
  -> Sequence Diagramm, Activity Diagramm, Communication Diagramm
- Implementation View: statische Software Komponenten: Fokus Implementierung
  -> Component Diagram
- Deployment View: Ausführbare Applikation: Fokus System Engineers
  -> Deployment Diagram
- Use Case View: gemeinsamer Nenner

Design Prinzipien:

- Coupling and Cohesion: Ziel: Gleichgewicht, Sequence- und Collaboration Diagramm
- System Control: Share: Verteilt
  Fork: Zentral
- Abstraction: Concept vs Value; Ignorieren von Details
- Decomposition und Modularisierung
- Encapsulation / Information Hiding
- Trennung Interface und Implementierung

Implementierung und Integration

-> Standardisierung: Namenskonventionen, Formatierungsrichtlinien, ...

-> Traceability: Zeitliche Traceability
  Horizontale Traceability: Beziehung zw. Entwicklungsartefakten
  Vertikale Traceability: Beziehung zw. Phase und Artefakttyp

-> Interne Standards, Übergreifende Standards

-> Integrationsstrategien:
- Big Bang Integration: alle Komponenten gleichzeitig
- Top-Down Integration: ausgehend von Business Cases
- Bottom Up Integration: ausgehend von Hardware
- Build Integration: Strukturiert entsprechend den Business Cases

Qualitätssicherung und Testen

→ Validation: Prüfung in Hinblick auf Spezifikation (Komponentenebene)
→ Validierung: Prüfung gegen Anforderungen (Nutzerebene)

→ Testtechniken: Unit Test: Testung Komponente
Integrationstest: Verbindung zwischen Komponenten
Systemtest: Übereinstimmung zwischen funktionalen und nichtfunktionalen Anforderungen
Regressionstest: Testen von geänderten Komponenten
Akzeptanztest: Übereinstimmung mit Anforderungen
Installationstest: Fehler während Installation

→ Testmethoden: Black Box Test
White Box Test

↳ Äquivalenzklassen
↳ Grenzwertanalyse

→ Testprotokollierung

Betrieb und Wartung

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

Corrective: Bugfix
Adaptive: Hardwareänderung
Perfective: Verbesserung, Effizienz
Preventive: Ergänzung Dokumentation

→ Sichten auf Wartung: Activity View: Änderung nach Instruktionsatz
Process View: Schritte zur Durchführung
Phase-Oriented View: Wartungsphase beginnt mit Auslieferung und endet mit Stilllegung

4 Technik und Werkzeuge

Source Code Management

-> Entwicklung: Revision Control Systeme
↓
Centralized Version Systeme
↓
Verteilte SCM

-> Zentralisierte vs. Verteilte Systeme

Build Management: kompiliert Applikation automatisiert erzeugen

-> Convention over Configuration (maven)
-> Dependency Management
-> Reporting und Dokumentation

Continuous Integration: Server-basierte Integration und Ausführung von Tests

↳ Code Quality, Kompilierung, Tests, Packaging, Tests, ...

Komponentenorientierte Software-Entwicklung

-> Direkter Aufruf
-> Interface
-> Komponenten
-> Services

-> Verbinden von Komponenten:
↳ Inversion of Control: Abhängigkeiten in Container verwalten
Dependency Injection

↳ Constructor Injection
↳ Setter Injection

Kommunikation im Team

-> Synchron: Face To Face, Telefon, IM, VOIP

-> Asynchron: Email, Wiki, CMS, Mailing Liste, Forum, Issue Tracker

5 Softwareprozesse

→ Vorgehensmodelle setzt auf bestimmte Kriterien zugeschnitten
↳ Dabei meist Definition der Anforderungen bis Softwareabnahme etc

Traditionelle Ansätze

→ Wasserfall Modell: lineare Life-Cycle, Schwerpunkt auf Dokumentation
→ V-Modell mit QS Methoden
→ V-Modell XT: Erweitert
→ Rational Unified Process: Inkrementell und Iterativ

Agile Ansätze

- Manifest mit 12 Prinzipien

→ Extreme Programming
→ SCRUM

Prozess Tailoring: Anpassung von Software Prozessen an aktuelle Projektgegebenheiten

Process Customization: Standardisierung: Anpassung an Unternehmensstandards, Projektstandards

↳ Vorteilhafte Form des Tailoring

→ Softwareprozesse / Vorgehensmodelle orientieren sich an Software Life-Cycle Prozess
... entspricht einer konkreten Strategie zur kontrollierten Durchführung eines Projekts

6 Modellierung von Anwendungsszenarien

-> Softwareentwicklung erfordert Herstellung konsistenter Schichten auf
- Anforderungen
- Systementwurf

-> Modelle: Überblick behalten, gemeinsame Notation

-> Prinzip der Abstraktion: Information Hiding, Vererbung

- UML Modell: Menge von Modellelementen
  -> Klassen, Attribute, Interaktionen, ...
- UML Diagramm: Sicht auf ein Modell
  -> Use Case Diagram, Class Diagram, ...
- Modellierungswerkzeug: Implementierung die UML
- Statische Aspekte UML: Anwendungsfälle
  Klassen
- Dynamische Aspekte UML: Sequenz
  Aktivität
  Zustandsautomat

-> Prozess und Datenfluss Analyse -> IDEF0
  -> Modellierung des Problembereichs
  -> Datenfluss (Daten-Token), Kontrollfluss (Kontroll-Token)

-> Modellierung: Konzeption -> Anforderungen -> Entwurf

- Modelle, wo?: Datenbanken: ER Diagramm
  Business Logic: Daten und Kontrollfluss
  Schnittstellen

  Qualitätssicherung, Qualitätsdefinition
  Test Driven Development

- Modelle, wie? Anwendungsfalldiagramm, Daten- Kontrollflussdiagramm
  werden Top-Down verfeinert

  -> Jede Phase erfordert andere Abstraktionsebene

Datenmodellierung (ER): Schema einer Datenbank
Entitäten, Attribute, Schlüssel, Relationen
Kardinalität-, Integritätsbedingungen

Verifikation und Validierung: Black Box Tests
Unit Tests via Mocking

Aktivitätsdiagramm: Daten- und Kontrollfluss

↳ Aktivität: spezifiziert benutzerdefiniertes Verhalten
- gerichteter Graph
- Aktionen, Kontroll- und Datenflüsse
- Parameter, Vor- und Nachbedingungen

↳ Testfälle: Alle Knoten und Kanten
- Wesentliche Pfade
- Variation der Eingabe- / Ausgabedaten

Zustandsdiagramm: Mögliche Folge von Zuständen eines Modellelements

↳ Zustände, Zustandsübergänge, Ereignisse, Aktivitäten

7 Qualitätssicherung

⇒ Qualität: Eignung zur Erfüllung vorgegebener Anforderungen
IEEE: Korrektheit, Effizienz, Verwendbarkeit, Testbarkeit, Wartbarkeit

↳ termingerecht im Rahmen des Budgets erstellt
↳ für Anwender verwendbar
↳ für Programmierer verständlich und änderbar
↳ für Betreiber effizient und administrierbar

⇒ Qualitätssicherung: Validation und Verifikation in jeder Phase

⇒ Testbarkeit: testbare Anforderungen
verfolgbare Entwicklung der Anforderungen in testbare Produkte

⇒ Basis: Anwendungsszenarien, Datenmodelle, Datenflussmodelle, Kontrollflussmodelle, Zustandsdiagramme, Sequenzdiagramme, Testspezifikation

Reviews: strukturierter Analyse- und Bewertungsprozess
qualitative Beurteilung von Produkten oder Prozessen

⇒ verwandte Tätigkeiten: Inspektion, Walkthrough, Audit

⇒ Rollen: Moderator, Leser, Gutachter, Schreiber, Autor

⇒ Phasen: Planung, Initialisierung, Vorbereitung, Reviewsitzung, Sammlung im Team, Reviewbericht, Nacharbeit

↳ Produkt reviewen, nicht Autor
Diskussion neutral und kurz halten
Reviews im Nachhinein bewerten

Schriftliche Aufzeichnungen,
gute Vorbereitung aller Teilnehmer
Arbeitsplan verwenden

Inspektionsplanung: Anforderungen → Modelle → Implementierung
Anforderung → Qualitätsdefinition (Modell) → Qualitätssicherung
(Review, Test)

Anwendungsfälle: Software Requirement Review, (Hardware + Operationing)
Klassendiagramme: Preliminary Design Review (Operationing + Entities)
Datenbankrelationen: ER-Modell systematisch durchgehen
Kontrollflussdiagramm: Alle Anwendungsszenarios sind abgedeckt
Zustandsdiagramm: — " —

⇒ Reviews mit Kunde:

Software Requirements Review
Preliminary Design Review
Critical Design Review
SA-Process Review

⇒ Reviews ohne Kunde:

Management Review
Inspection
Code Walkthrough
Technical Review

Testen

-> Testarten, Testziel (Funktion, Kont), White/Black Box -> Testart

-> Rollen: alle Teammitglieder sind Testverantwortliche
Technische Mitglieder: testbarer Code
Tester: auf TDD achten, System-Test durchführen

-> Normal-, Sonder-, Fehlerfälle testen
↳ alle Knoten und Kanten in einem Kontrollflussgraphen

-> Äquivalenzklassenanalyse: Bestimme Untermengen von Input-Daten, welche gleiche Auswirkungen haben

↳ Anzahl Testfälle sinkt
↳ Untermengen von Eingabewerten (explizite Parameter, implizite Systemzustand)

↳ Grenzwertanalyse: Grenzen der Äquivalenzklassen

-> Value-Based Testing
-> Requirement-Based Testing
-> Risk-Based Testing

-> Testfallbeschreibung: Bezeichnung
Vorbedingung
Eingabewerte
Aktionen
Erwartetes Ergebnis
+ Tatsächliches Ergebnis
+ Bewertung

- Unit Testing: Dokumentation von Klassen, Testen gegen Spezifikation

- Modultests: Modul: dokumentierte Spezifikation, explizite Integration ins Gesamtsystem, getrennt von anderen Modulen testen

- Integrationstest: Testet Interaktion zwischen Modulen

-> Black Box: Bezug auf Spezifikationen -> Unit Tests (TDD)
Anforderungsüberdeckung
Partitionierung der Eingabedaten

-> White Box: Bezug auf Struktur von Code bzw. Modellen -> Modul-, Systemtests
Überdeckung von Knoten, Kanten und Pfaden
Partitionierung von Daten für die Bedingungsauswertung

- Test Driven Development: Think: Test spezifizieren
Red: Test implementieren
Green: Komponente implementieren
Refactor: Test sollte nie wieder fehlschlagen

- Best Practices: kleine Programme / Methoden
Objekte so testen, wie sie verwendet werden
Test Cases sind Living Documents
Eigenständig ausführbare Klassen
Bottom Up-Testing

- Frameworks: JUnit, .NETUnit

8 Persistenz

→ Anforderungen

↳ Datenstrukturen : Binär, Semi-Strukturiert, Strukturiert (DB), ..
↳ Konzepten : Anzahl Anwender, Leistungserwartung, erwartete Größe
↳ Zuverlässigkeit : Integrität, Verfügbarkeit, Backup-Szenarien
↳ Legacy : Aufwand Einführung / Administration, Vorhandene Daten
↳ Aufwand : Entwicklungszeit, Administration, Wartung

File Persistence

Relationale Datenbanken : Tabellen, Zeilen, Spalten
Typsicht
Indexing
Transaktionen (ACID)

↳ Additional Features
↳ Enterprise Features

- Embedded Datenbanken : Serverless, Leichtgewicht
- Zugriff : SQL, Low-Level Interfaces : JDBC, ODBC, Utility Libraries
- Objektrelationales Mapping : → Impedance mismatch

↳ manuell : JDBC
↳ Data Mapper : MyBatis
↳ Full blown OR Mapper : JPA

Objektorientierte Datenbanken : Verwendung des Domänenmodells
kein Mapping notwendig
Transaktionen, Performance, Clustering

→ db4o

XML Datenbanken : Hierarch. Modell für XML Dokument
Verwaltung von Informationen in XML Format

- Native XML Datenbanksysteme : Speichern physische Struktur
Zugriff mittels XML Technologien
→ Xpath, X query, XSLT
Strenge Trennung

NoSQL Datenbanken: Nicht-relationales Ansatz
kein vordefiniertes Datenmodell
Horizontale Skalierung

↳ ACID nicht im Vordergrund:
↳ Fokus auf Skalierbarkeit
↳ Anwendungsfall bestimmt Datenstruktur, API

→ CAP-Theorem: Consistency – Partition Tolerance – Availability

- Key-Value Datenbanken: Zugriffsprüfung
Limitierung bei komplexen Anfragen
→ Webserver

- Spaltenorientierte Datenbanken: Speichert Daten spaltenweise
→ Minimierung von Null-Zellen
→ Cassandra

- Dokumentenorientierte Datenbanken: Hash Table ähnlich
→ einzige ID Feld

↳ Dokumente: Strukturierte Daten in unterschiedlichen Formaten
key-value Paare, JSON, XML

↳ Verschachtelte Strukturen

→ Mongo DB, Couch DB

- Graphendatenbanken: Benutzung von Graphen zur Darstellung und Speicherung von Daten

↳ Knoten, Kanten / Beziehungen
↳ Navigation über Kanten

→ Neo 4 J

Polyglot Persistence: multiple data storage technologies
based on the way data is being used

Fokus: Availability – Consistency: RDBMS
Availability – Partition Tolerance: Cassandra
CouchDB
Riak
Consistency – Partition Tolerance: HBase
MongoDB
Redis

9 People Management

Motivation: extrinsische Motivation: äußere Zwänge / Maßnahmen
intrinsische Motivation: von Aufgabe ausgehender Anreizen bedingt

→ Motivationstheorien

↳ Prozesstheorien: Erklärung wie Verhalten gesteuert wird, warum bestimmtes Verhalten zur Zielerreichung gewählt wird

- Anspruchsniveau
- Persönliche Einstellung
- Subjektive Wahrscheinlichkeit

} Einflussfaktoren

↳ Gleiche Anreize führen zu unterschiedlichen Einflussfaktoren und somit zu unterschiedlicher Motivation
↳ Jede Erfahrung beeinflusst Motivationsprozess

↳ Inhaltstheorien: Beschreiben Bedürfnisse, Antriebe, Ziele der Menschen

- Monothematische Theorien: einzelnes Motiv ausschlaggebend
- Polythematische Theorien: mehrere Grundmotive bestimmen Verhalten

↳ Motivationspyramide (Maslow): Physiologische Bedürfnisse
Sicherheitsbedürfnisse
Soziale Bedürfnisse
Achtung
Selbstverwirklichung

Produktionsfaktor Mensch: Produktivität
Fluktuation
Maßnahmen der Leistungsbereitschaft

→ Faktoren Produktivität: + Teamgeist
+ Arbeitsplatz
↳ U-Faktor

- Programmiersprache
- Berufserfahrung
- Anzahl Fehler
- Gehalt

Auswahl von Mitarbeitern: Job Spezifikation
Job Inhalt profile
Bewerbungen einholen
Auswertung Bewerbungen
Auswahl Mitarbeiter

- Portfolio
- Eignungstest
- Anforderungen
- Interviews (mehrere Bewerber)

Team Management: Don't Dos: Reaktives Management
Burnout
Physikalische Trennung

Qualitätsreduktion
Schuldzuweisung (Unverständnis)
Qualitätskontrolle

10 Software Patterns

→ Managing Complexity: Standardization
Modeling
Transformation
Experience
Abstraction
Decoupling
Decomposition
Classification

→ Pattern: ... solution to a problem in a context

→ Elements: Meaningful name
Motivation and Problem statement
Context
Solution: Structure, Participants, Collaboration, Consequences, Implementation, Examples

↳ Common Vocabulary
↳ manage complex systems
↳ improve documentation
↳ minimize development time

↳ do not lead to dead code ends
↳ deceptively simple
↳ suffer from pattern overload

→ Classification: Architectural: structure, communication of systems
Design: structure, relations of classes
Idioms: low level details, language specific
Protopatterns: Particular case
Anti Pattern: commonly used ineffective technique

→ Types: Creational: initializing, configuring objects
Structural: decoupling classes and objects
Behavioral: dynamic interaction

Fundamental Patterns

- Interface : distinct between behaviour and concrete implementation
- Delegation : Outsource functionality into new class
- Immutable : Once created state of instance must not change
  → initialize in constructor, only getter

Creational Patterns:

- Singleton
- Factory : Method in derived class creates associates
- Abstract Factory : Build related Objects without specify concrete class
- Builder : Factory for building complex objects in different variants
- Prototype : Factory for cloning new instances from prototype instance

Structural Patterns

- Facade : Simple interface for a sub-system
- Adapter : Adapts an interface to match something needed (wrapper)
- Proxy : Extends delegation, provides interface functionality
- Bridge
- Composite
- Flyweight

Behavioral Patterns

- Observer: Notify on subject change
- Decorator: extends an object transparently
- State: Object behavior depends on internal state
- Strategy: Vary algorithms independently
- chain of Responsibility: Delegate request to responsible service provider
- Iterator: Access collection of elements sequentially
- Command: Store all information to call a method at a later time
- Mediator: Coordinate interactions between associates
- Memento: capture and restore object states

SEPM VORGEHENSMODELLE

Wasserfallmodell: nicht iterativ

Umsetzung des Software Life Cycle
jede Phase hat definierte Start- und Endpunkte mit definierten Ergebnissen
Schwerpunkt auf Dokumentation (Phasenergebnis)

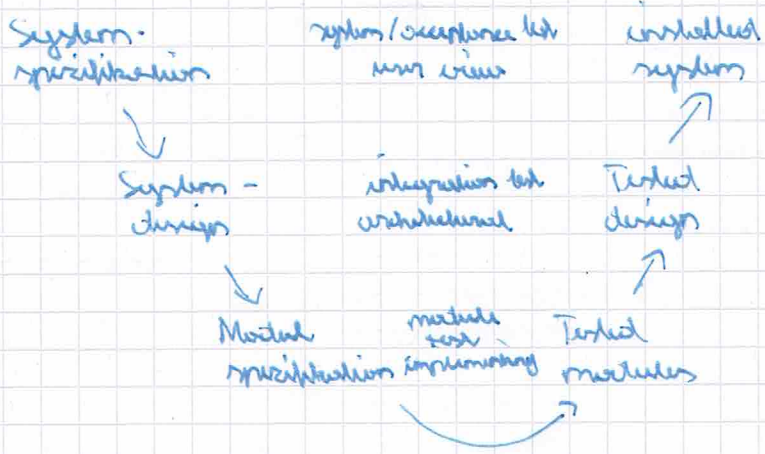
→ Wasserfallmodell mit Rücksprung führt iterative Aspekte ein

+ Strikte Trennung der Phasen
+ Klare Anforderungen stabil, effizient
+ Klares Team

- Keine Phasentrennung umsetzbar
- Unflexibel gegenüber Änderungen
- Fehler werden evtl. erst spät erkannt

V-Modell mit QS

Spezifikations- und Implementations/Testphasen stehen sich gegenüber. Bei Fehler in der Testphase muss in die jeweilige Spezifikationsphase zurückgesprungen werden.



+ Verschiedene Abstraktionslevel (User, Architektur, Implementierung)
+ Frühe Fehlererkennung durch Reviews

- Klare Systemanforderungen notwendig
- Hoher Dokumentationsaufwand

⇒ Große Projekte im öffentlichen Bereich

→ Qualitätsplan von System Spezifikation
→ Reviews und Testplan nach jeder Spezifikationsphase

V-Modell XT

Verbesserung in Bereichen Anpassbarkeit, Interoperabilität, Skalierbarkeit / Erweiterbarkeit des V-Modells
→ Standard im deutschen öffentlichen Bereich

Produkte stehen im Mittelpunkt.
→ Definierte Rollen mit Verantwortlichkeiten

Vorgehensbausteine sind modulare Elemente
beinhalten Produkte, Rollen, Aktivitäten

Entscheidungspunkte definieren Zeitpunkt, an dem eine Fortschrittsentscheidung getroffen wird

Projektdurchführungsstrategien definieren Reihenfolge von Projektdurchführungsschritten

Projekttypen werden eingeteilt nach:
Projektgegenstand (Hardwaresystem, Softwaresystem)
Projektrollen

⇒ Systemerstellungsprozess des Auftraggebers
des Auftragnehmers
in - house
Einführung + Pflege eines organisationsspez. Vorgehensmodells

In Entscheidungspunkten müssen definierte Produkte vorliegen.

Rational Unified Process: anwendungsfall- und risikoorientierte Architektur im Zentrum der Planung

4 grundlegende Phasen:
- Inception: Anforderungsanalyse, Machbarkeitsstudie, Risiken
- Elaboration: Architekturprototyp, Anwendungsfallbeschreibung
- Construction: Implementierung
- Transition: Auslieferung an Kunde

Jede Phase besteht aus ein oder mehreren Iterationen mit 9 Arbeitsschritten

- 6 Entwicklung Workflows
- 3 Supporting Workflows

+ Werkzeugunterstützung
+ Real-World Szenarien
+ Vorgefertigte Lösungen mit erforderlichen Anpassungen

- Hohe Komplexität
- Dokumentationsaufwand

Agile Vorgehensmodelle

Individuals and interactions	over process and tools
working software	over documentation
customer collaboration	over contracts
Responding to change	over following a plan

Scrum: empirisch, inkrementell, iterativ

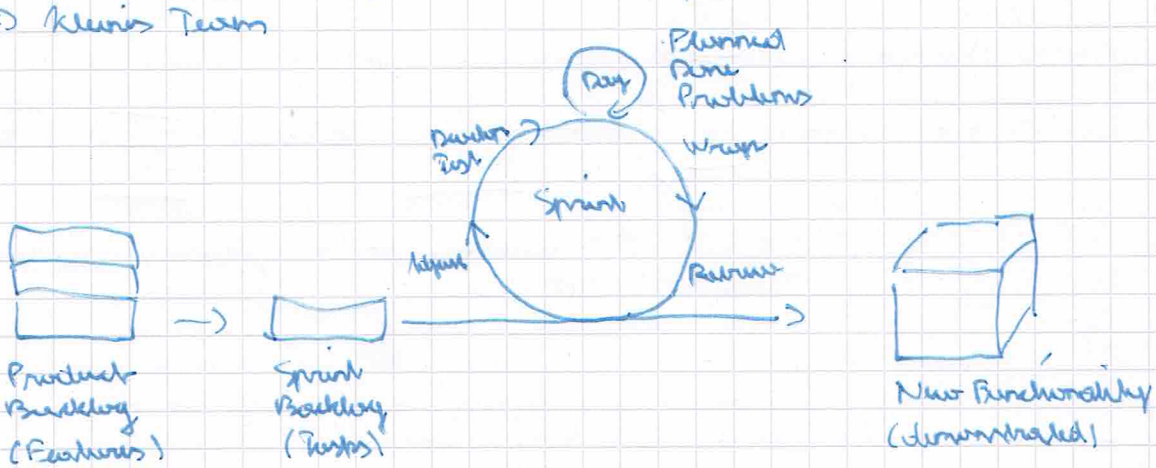
Annahme: Entwicklungsprozesse zu komplex um komplett zu planen
-> Zwischenergebnisse schaffen und diese bewerten

Ziel: schnelle kundengerechte Entwicklung

Anforderungen werden aus Benutzersicht formuliert und in Product Backlog gelegt.

Anforderungen werden inkrementell in Sprints entwickelt.
Nach einem Sprint soll ein funktionierendes Produkt vorliegen.
Dieses wird überprüft und im nächsten Sprint weiterentwickelt

=> Steuerung: Prozesse, Methoden, Rollen
=> Innerhalb von Sprint keine Eingriffe von außen
=> Kleine Teams



Anforderung aus Benutzersicht in Product Backlog.

Auswahl von Features für Sprints, Zerlegung in Tasks.
Implementierung, Tests, Review von Tasks.
Jeden Tag Daily.
Funktionierendes Teilprodukt

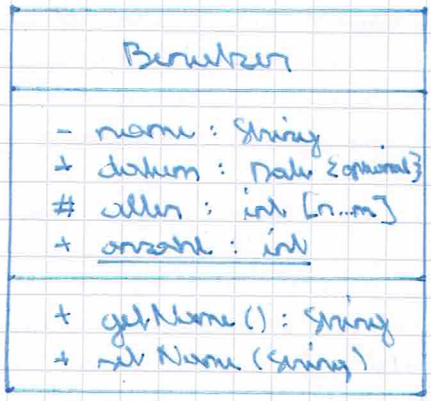
+ Hohe Flexibilität
+ Hohe Transparenz
+ Dokumentationsaufwand

- Wenig klare Regeln
- Kleine Teams
- Kein Gesamtüberblick

SEPM UML

Klassendiagramm

... strukturellen Aspekt eines Systems auf Typebene in Form von Klassen, Interfaces, Beziehungen



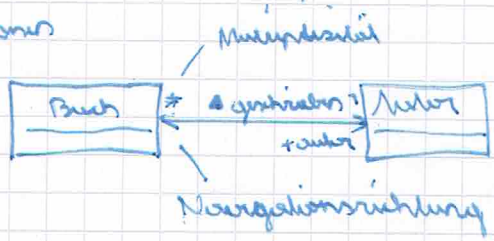
Klassenname

Attribute

Operationen

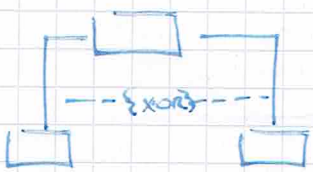
private
public
protected
static

Assoziationen



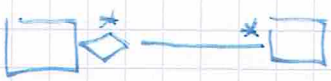
=> auch als Attribut möglich

+ autor : Rolle



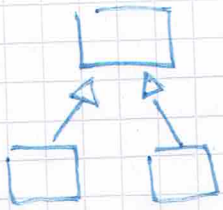
Exklusive Assoziation

Assoziationsklasse

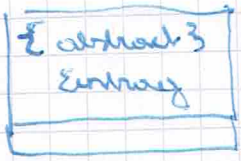


Aggregation

Komposition

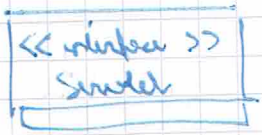


Generalisierung



abstrakte Klasse

ordnet, umgibt



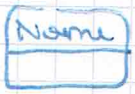
Interface

Zustandsdiagramm

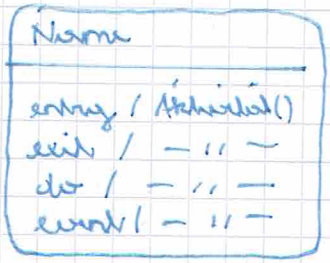
... beschreibt mögliche Folge von Zuständen eines Modellelements
→ während gesamten Lebenszeit / Ausführung einer Operation

=> Zustände / Zustandsübergänge, Ereignisse, Aktivitäten

Zustand: Start (Pseudo)



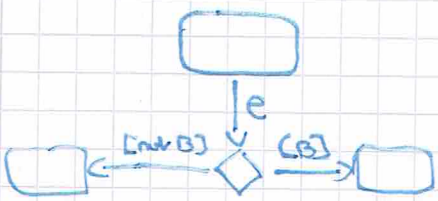
Endzustand X Terminierungsknoten



entry: bei Eingang in Zustand
exit: bei Verlassen von Zustand
do: während im Zustand

event: Wenn Ereignis event auftritt

Zustandsübergang:



e ... Ereignis
[B] ... Guard

default event: da ausgehenden

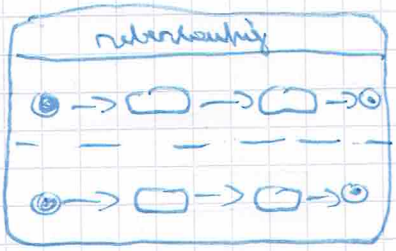
=> Ereignis, Bedingung, Aktivitäten auf Zustandsübergang möglich

Ereignis (Argumente) [Bedingung] / Aktivität

Parallelisierung

Synchronisation

Komplexe Zustände



Aktivitätsdiagramm

Aktivität ... gerichteter Graph: Knoten: Aktionen
Kanten: Kontroll- / Datenfluss

... Beschreibung des Ablaufs eines Anwendungsfalls, durch Spezifikation von Kontroll und Datenfluss zwischen Arbeitsschritten

Aktivität beschreibt eines Anwendungsfall / einen Ablauf im System.

=> Aktionen, Kontrollfluss, Datenfluss

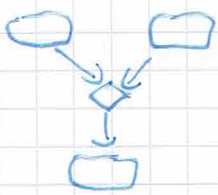
Aktion: atomar, können Eingabe / Ausgabewerte haben.
-> Spezielle Notation für bestimmte Aktionsarten

Kontrollflusskanten: Kontrollabhängigkeit zwischen Vorgänger und Nachfolge-Aktion.

Datenflusskanten: Objektfluss. Datenabhängigkeit

- Startknoten
- Aktivitätsendknoten
- Aktionsendknoten
- Aktivitätsendknoten

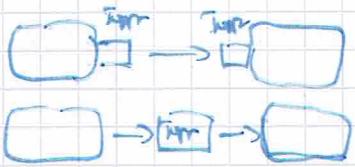
Entscheidungs-knoten



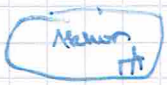
Vereinigungs-knoten

Parallelisierungs-knoten

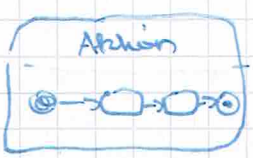
Synchronisations-knoten



Objekt-knoten



Schachtelung von Aktionen



SEPM FRAGEN

Traceability:

Nach- / Rückverfolgbarkeit einer Information durch einen gesamten Entwicklungszyklus

Änderungsverfolgung: Anforderung bis Implementierung verfolgbar

-> Nachvollziehbarkeit der Änderungen
-> Information in Header Block

Zeitlich: zeitliche Nachvollziehbarkeit (versch. Releases)
Vertikale: Abhängigkeiten, Beziehungen innerhalb eines Phase und eines Artefakttyps (System - Subsystem)
Horizontale: Beziehungen zwischen unterschiedlichen Artefakttypen (Anforderung - Implementierung)

Äquivalenzklassen

Bilde Menge von Eingabewerten (explizite Parameter, implizite Systemzustand).

Zerlege in Mengen, welche gleiche Ergebnisse / Auswirkungen haben -> Äquivalenzklassen

Grenzwertanalyse:

Grenzen der Äquivalenzklassen werden als Vertreter - repräsentativen Auswahl.

Anforderungskategorien:

Funktionale: Welche Funktionen sollen umgesetzt werden?, Wie soll sich das System in definierten Situationen verhalten

Nicht-funktionale: Kosten, Performance, Usability, Sicherheit

Randbedingungen: Plattformen, Entwicklungsumgebung

Prozessbedingungen: Rahmenbedingungen: Ressourcen, Dokumentation

Test Driven Development

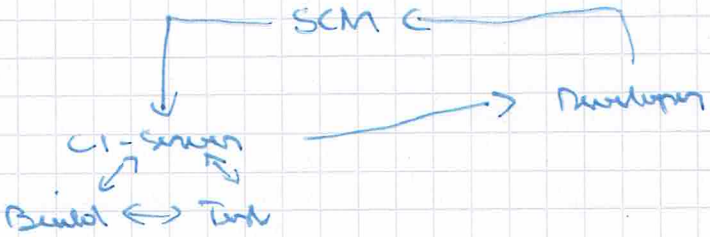
Testfall vor Implementierung aus Spezifikation

Think: Testfall mit Spezifikation definieren
Red: Test implementieren
Green: Implementierung der Funktionalität
Refactor: Funktionalität bleibt erhalten -> Test sollte nie weiter fehlschlagen

Continuous Integration

Automatisierung: Build, Tests, Deployment, Integration

Server basiert CI: Source code Repository



Integrationsstrategien:

Big Bang: Gleichzeitige Integration aller Komponenten

Top Down: Schrittweise Integration ausgehend von Business Cases

Bottom up: Ausgehend von Hardware

Build Integration: Entsprechend der Business Cases

Inversion of Control

Dependency Injection: Abhängigkeiten werden in einem Container verwaltet und werden in Komponenten injiziert. Komponenten wissen nichts darüber

=> Don't call us, we call you

Projektauftrag:

... schriftliche Vereinbarung zwischen Auftraggeber und -nehmer, ein Projekt zu bestimmten Bedingungen durchzuführen

... umfasst alle wichtigen Daten für zukünftige Projektdurchführung

=> Projektname
Beschreibung
Rollen und Verantwortlichkeiten

Lieferumfang
Anforderungen (+ nichtfunktional)
Nutzen - Ziele, Abgrenzung

Zeit-, Kostenplan
Risiko-, Umweltanalyse
Komponentendiagramm

Projektmanagement

... Methoden und Verhaltensweisen zur effizienten Steuerung von besonderen Aufgabenstellungen

=> Planung, Kontrolle, Steuerung
=> Projektablauf, Arbeitsstruktur, Informationswesen

System control:

... von übernimmt Kontrolle

Stair: Verteilte Kontrolle

Fork: Zentrales Objekt kontrolliert gesamten Use-Case

Projektsteuerung: Steuerung durch Soll - Ist - Vergleich

=> Meilensteine,
Balkendiagramme (Gantt)
Burn Down chart

Testdurchführung:

=> Qualität
Quantität
Entwicklungsstufen
Kosten

Projekt:

... Einmaliges Vorhaben mit definiertem Anfang, Ende und mehreren beteiligten Personen

Abgrenzbar: Erwartetes Ergebnis (Funktionsumfang, Lieferumfang, Qualität)
Ressourcen (Geld, Personal)
Zeitrahmen

Software Lifecycle:

... Basiskonzept für Vorgehensmodelle

- Anforderungen + Spezifikation
- Planung
- Design und Entwurf
- Implementierung und Integration
- Betrieb und Wartung
- Retirement

Design Principles:

- System Control
- Information Hiding
- Modularisierung
- Interface / Implementierung

4+1 View Model

- Logical View: Funktionale Anforderungen, Use Case
- Process View: Nicht-Funkt. Anforderungen, Sequence
- Implementation View: statische Komponenten, Component
- Deployment View: ausführbare Applikation, Deployment

- Use-Case-View: gemeinsamer Nenner Anwendungsfälle, Szenarien

Wartungskategorien:

	Correction	Enhancement
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

- Preventive: Ergänzung Dokumentation
- Perfective: Verbesserung Effizienz
- Corrective: Bugfix
- Adaptive: Hardware änderung

Wartungstechniken

Reengineering: Überprüfen und Überarbeiten von Code

Reverse Engineering: Analyse hinsichtlich auf Komponenten Erstellung von Modellen aus Code

Vorgehensmodell:

.. orientiert sich am Software Lifecycle Prozess
.. entspricht konkreter Strategie zur Erstellung eines Software Produkts

Reviews:

mit Kunde:
- Software Requirements Review
- Preliminary Design Review
- Critical Design Review
- In Process Review

ohne Kunde:
- Management Review
- Technical Review
- Code Walkthroughs
- Inspection

Rollen: Moderator, Autor, Leser, Gutachter, Schreiber

Tests:

Test Stufen:
- Unit
- Modul
- System

Test Art
- White box
- Black box

Spezifikation: Bezeichnung, Voraussetzung, Eingabewerte, Aktionen, Erwartetes Ergebnis, Ergebnis, Bewertung

Prozess Tailoring:

... Anpassung von Software-Prozessen an aktuelle Projektgegebenheiten

Prozess Customization

... Anpassung von Vorgehensmodellen an
- unternehmensspezifische Gegebenheiten
- Projektstandards (zB Web-Applikationen)