

TB 2 - Softwaresysteme

1. Vorgehensmodelle

Als Grundlage SDLC

SDLC

Software Development Life Cycle

Gute Planung führt zu geringeren Betriebs- & Wartungskosten

Was

1. Idee & Projektanstoß
2. IST-Erhebung
 - was gibts bereits?
 - wie sehen die Systeme aktuell aus?
3. Anforderungen erfassen
 - **funktionale** (was soll SW können?) & **nicht-funktionale** (Performance, Sicherheit, ...)

Wie

4. System & Komponentenentwurf
 - Systemarchitektur
 - technische Spezifikation
 - Schnittstellen
 - ...

Implementierung

5. Implementierung
6. Komponententests
7. Integrations & Systemtests
8. Abnahmetests (hält System Spezifikation)

Betriebsnahme - Deployment/Release - Außerbetriebnahme alter Systeme - Betrieb & Wartung

Phasenmodelle

Entwicklung verläuft sequentiell & schrittweise

Traditionelle Modelle

Beispiele: - Wasserfallmodell - Spiralmodell

Wasserfallmodell

- alte Herangehensweise

- kommt aus anderen Disziplinen & klassischen Projekten
- eher weniger bei SW-Projekten
- wenn eine Phase abgeschlossen ist gibt es kein zurück mehr
- erst wenn vorherige abgeschlossen ist kann nächste Phase beginnen
- Kosten bei fixen Anforderungen leicht abschätzbar
- **Dokumentgetrieben** (nach jeder Phase muss ein Dokument vorliegen)
- **top-down**
- nicht mehr anwendbar bei SW (Anforderungen können sich schnell ändern)
- Planungsfehler erst spät ersichtlich (late design breakage)

Spiralmodell

- es gibt Phasen die sich wiederholen
- Es wird in Zyklen gedacht

Schritte:

1. Ziele definieren für nächsten Zyklus
 2. Risikoanalyse & Prototyping
 3. Durchführung und Evaluation
 4. Planung der nächsten Phase
- frühzeitige Evaluierung
 - Prototypische umsetzung
 - Risikominimierung

V-Modell

- verfolgt Test Driven Development
- Dokumentorientiert
- Fokus auf Qualitätssicherung

linke Seite

- Etappen des SDLC
- vor Durchführung Tests ausdenken & Implementierung Evaluieren

rechte Seite

- Tests für jew. Entwicklungsschritte
- auf technischer Ebene = funktioniert System überhaupt?
- auf benutzer Ebene = Bieter das System dem Nutzer den gewünschten Nutzen
- utility/warranty

RUP - Rational Unified Process

- erster Schritt in Richtung agile Modelle

- basiert auf UML (beschreibt auf allen Ebenen Projekt mit Hilfe von UML-Diagrammen => Ausgehend von UseCases)
- Architekturzentriert
- in jeder Phase werden Workflows durchlaufen
 - Business Modelling
 - Requiring (Anforderungen erheben)
 - Analysis & Design (Grobspezifikation)
 - Implementation
 - Tests
 - Deployment
- Supporting Workflows
 - Configuration & Change Management = wie reagiert man auf Anforderungsänderungen
 - Project Management
 - Environment = Arbeitsumgebung schaffen
- Aufwand für jeden Workflow ist abhängig von der aktuellen Phase
- in jeder Phase kann es 1 bis mehrerer Iterationen geben die jew. ein Produktinkrement liefern
- Elaboration braucht am meisten Aufwand & Zeit
- Late Design Breakage ist sehr unwahrscheinlich

Phasen:

1. Inception
 - Anforderungen identifizieren
 - Wirtschaftlichkeit
 - Risikoanalyse
 - Machbarkeitsprüfung
 - Validierung mittels ersten Prototypen
 - **LCO** = Lifecycle Objective Milestone
2. Elaboration
 - Architektur erstellen
 - technische Spezifikation) = Lifecycle Architecture Milestone (= Point of no return)
3. Construction
 - Umsetzung/Implementierung
 - **Initial Operational Capability Milestone** = fertiges System
4. Transition
 - Übernahme von Entwicklungs- auf Produktionsumgebung
 - Testen
 - Inbetriebnahme
 - **Product Release**

Agile Modelle

- Anforderungen sind veränderlich (daher sind Kosten schwer einschätzbar)
- Wenn Phasen strikt eingehalten werden passt finales Produkt nicht
- flexiblere Planung

Agiles Manifesto:

- enthält wichtige Grundsätze für agile Vorgehensmodelle
- **Individuals and interactions** over processes and tools
 - Selbstverantwortung & Motivation
 - Zusammenarbeit
- **Working software** over comprehensive documentations
 - Erfolg an Produkt messen und nicht an dokumentation
- **Responding to change** over following a plan
 - Anforderungsänderungen berücksichtigen & willkommen heißen
- **Customer collaboration** over contract negotiation

SCRUM

Rollen:

- Product Owner
 - definiert User-Stories (Anforderungen mit Akzeptanzkriterien) & filtert wichtigste heraus
 - verwaltet Product Backlog (enthält User-Stories)
- Team
 - setzt Anforderungen um
 - umsetzung in Sprints (enthält nun unveränderliche User-Stories die umgesetzt werden)
 - arbeitet autonom und selbstorganisiert
- Scrum Master
 - hilft die Umsetzung des Modells

Sprints:

- Aufwandsschätzung vor Sprint mithilfe von **Planning Poker**
- Sprints dauern 2-4 Wochen
- am Ende ein Potentially Releasable Product
- Daily Scrum Meetings (welche Tasks gestern erledigt worden sind & was wird heute erledigt)
- Sprint Review & Retrospective Meetings

User-Story:

- verfolgt Muster “Als Kunde will ich folgenden Nutzen erreichen”

Anforderungen:

- sollten INVEST Kriterien erfüllen
 - Independent

- Negotiable
- Valuable
- Estimatable
- Small
- Testable
- erst umsetzbar wenn Definition of Ready erfüllt ist
- fertig erst wenn Definition of Done erfüllt ist

Controlling mittels Burndown Chart:

- darstellung des Arbeitsfortschritts
- wenn User-Story fertig ist verringert sich der Wert der verbleibenden Story-Points
- man nähert sich idealem Burndown an

FDD - Feature Driven Development

Besteht aus 5 Stufen:

- Startup:
 - wie Inception & Elaboration
 - Überblick
 - Anforderungen
 - Wirtschaftlichkeit
 - Mögliches Modell
- Build A Feature List
- Plan by Feature
- Design by Feature
 - Design Package
- Build by Feature
 - Umsetzung & Testen

Extreme Programming

- ähnlich wie SCRUM
- versucht Änderungskosten gering zu halten
- Fokus auf Engineering Practices & ist sehr Praxisorientiert

YAGNI Prinzip = Klasse wird so implementiert, dass nur der Test erfüllt wird
(You Aint Gonna Need It)

Praktiken:

- Test-Driven Development
- Pair Programming
- Refactoring
- Continous Integration/Delivery
 - unterstützt Testen & Builds
 - Infrastrukturaktivitäten mittels Scripts lösen

- Starke Kohesion = 1 Klasse hat genau 1 Aufgabe
- Loose Coupling = minimale Bindung zwischen Klassen

Kanban

- kommt aus der Automobilbranche
- Verschwendung vermeidung
- Just-In-Time Konzept
 - nur Produzieren wenn Abnehmer etwas brauchen
 - Puffer dazwischen
 - Spart Lagerkosten
 - kein Überschuss

Vorgehen in Softwareprojekten:

- Kanban-/Taskboard spiegelt Schritte von SDLC wider (Next, Analysis, Development, Acceptance, Production)
- Jede Phase hat eine bestimmte Zahl = Work in Progress Limit (Wieviele User-Stories dürfen gleichzeitig in einer Phase sein)
- In jeder Phase werden User-Stories erledigt und danach in Done geschoben

2. IST-Erhebung

- Wie wird der aktuelle Systemzustand erfasst?
- Parallelen zur Anforderungsanalyse

Interview

- man redet mit Verantwortlichem
- bereitet Fragen vor (Standardisiert = Fragenkatalog; Nicht-Standardisiert = abweichend)
- weich/hart Interview (abhängig von Erst der Lage)
- offene/geschlossene Fragen
- qualitative Informationen
 - im Detail
 - Nachfragen
- wie sieht System genau aus?
- wie stellt sich Interviewpartner Verbesserungen vor?
- Auswertung ist Aufwendig und nur bei relativ wenigen Personen möglich
=> Dafür detaillierte Informationen erfassbar

Fragebogen

- bei einer größeren Zielgruppe
- eher standardisierte/erprobte Fragebögen verwenden
- Test bei einer kleineren Gruppe (Verständnis & Auswertbarkeit prüfen)

Beobachtung

- Mitarbeiter zuschauen
- wichtige Informationen protokollieren
- passiv/aktiv (aktiv = Fragen stellen)
- aufwendig

Selbstaufschreibung

- mittels Protokoll seitens Mitarbeiter
- ungenau
- erfordert Eigenverantwortung & wahrheitsgemäße Erfassung

Dokumentenauswertung

- bestehende Unterlagen untersuchen
- Problem bei Alter/Genauigkeit von Dokumenten

CRC - Karten

- Class Responsibility Collaboration
- Klassenmodell basierend auf Anwendungsfällen
- Class = Hauptwörter
- Responsibility = wer ist Verantwortlich
- Collaboration = wer ist involviert

3. Aufwandsschätzung

4. Geschäftsprozessmodellierung

GP = ist ein Ablauf

In einem Unternehmen gibt es:

- Ablauforganisation (Prozesse)
- Aufbauorganisation (Aufgaben, Verantwortlichkeiten, Stellen, ...)

Prozesse:

- Aufgaben (haben unter einander Verbindungen/Abhängigkeiten)
- Wer Verantwortlich
- Ressourcen
- Output
- haben ein betriebswirtschaftl Ziel

Modellierung:

- Prozesse abbilden
- Prozesse im Rahmen von PDCA verbessern

- auch Abteilungsübergreifend (siehe Prozessorientierung)

Probleme von Prozessen:

- zu lange Dauer
- zu hohe Kosten
- zu viele Fehler
- **Verbesserung durch Modellierung**

Prozessorientierung:

- Zusammenarbeit zwischen Abteilungen verbessern
- Prozesse in den Mittelpunkt
- Mehr abteilungsübergreifende Arbeit => hohes Maß an Verantwortung/Team-Skills
- Qualitätsverbesserung

Bestandteile:

- Ereignisse = Zustand in einem System
 - Prozess verläuft von Zustand zu Zustand
 - ZUstand wird mit Daten repräsentiert => Daten werden verändert

EEPK

Bestandteile:

- Funktionen (abgerundete Rechtecke) => aktiv benannt & enthalten verben
- Ereignisse (Sechsecke) => passiv benannt
- Informationsobjekte (Rechtecke)
- Organisationseinheit (Ovale)
- Kontrollflüsse (Pfeile)
 - Verzweigungen (AND; OR; XOR)
- Prozesswegweiser
 - Enthalten einen Prozessteile
 - zur Wiederverwendung
- Schleifen
 - XOR verknüpfung bei zurück kommen
 - irgendwo Abbruchbedingung
- Wartesituationen
 - warten in Funktion oder Ereignis AND verknüpfen

Regeln:

- GP muss mit Ereignis beginnen/enden
- Abwechselnd Funktion & Ereignis
- Ereignisse können keine Entscheidungen treffen (nur AND nach Ereignis)

BPMN

Business Process Model and Notation

- Standard der Object Management Group
- Tripple Ground Standard (CMN, DMN)

Bestandteile:

- Aufgaben
- Ereignisse
 - Startereignis (Einfache Linie)
 - * eingetretene (nicht ausgefülltes Symbol) = warten auf ereignis
 - Zwischenereignis (Doppel Linie)
 - * können auch an Aufgaben angehefter sein (unterbrechen/nicht-unterbrechend)
 - unterbrechend (durchgezogene Linie)= Token wandert Pfad von Ereignis weiter
 - nicht-unterbrechend (strichlierte Linie) = neues Token
 - * Nachrichtenereignis = löst Nachricht aus (asynchron)
 - * Zeitereignis = löst aus wenn bestimmter Zeitpunkt erreicht wurde
 - * Links = GP auf mehreren Seiten aufteilen
 - Endereignis (Dicke Linie)
 - * aufgerufene (ausgefülltes Symbol)= man wartet nicht
 - * Terminierungsendereignis = ganzer GP ist vorbei
 - Symbol bestimmt Art des Ereignis (sonst Blanko)
 - Terminierungsereignisse
 - * zerstören Tokens
- Gateways
 - X = XOR
 - * wird nicht warten
 - * wird nicht aufteilen
 - O = OR (parallelisierend)
 - + = And (zwingend parallelisierend)
 - **parallelisierend** = erschaffen mehrere Ablauftokens, Teilen GP auf (split/join)
 - **synchronisierend** = es wird auf Tokens gewartet
- Standardfluss
 - wenn mehrere Bedingungen nicht erfüllt wird dieser Ausgeführt
- Pool = Prozess
 - enthält mehrere Lanes
- Lanes = Organisationseinheit
 - kann Sublanes enthalten
 - keine überlappungen von Prozessen
- Sequenzflüsse
 - nur innerhalb eines Pools
- Nachrichtenfluss

- Signal zwischen Pools
- führen in eingetretene Nachrichtenereignisse / Aufgaben / ...
- Tokens
 - stellen den aktuellen Zustand/Punkt im GP dar
 - wenn alle Tokens Ende erreicht haben ist GP vorbei
- Kompensationsaufgaben
 - können zurückgerollt werden
 - auslösen mit angehefteten ereignis
- Aufgabensymbole
 - Schleifen
 - Mehrfachinstanziierung
 - Plus = enthält Teilprozess (wie Prozesswegweiser)

UML Aktivitätsdiagramm

- Veraltensdiagramm
- Modelliert Abläufe in Systemen

Bestandteile:

- Rahmen = Aktivität = Prozess
- Partitions = Swim Lane = Organisationseinheit
- Aktion
 - Accept Event Actions
 - * auf Ereignis warten
 - * Triggert weiteren Prozess
 - Send Event Action
 - * löst Nachricht/Event aus
 - * geht aber direkt & ohne Verzögerung weiter
- Knoten
 - Entscheidungsknoten = XOR
 - Fork = AND Split
 - Join = AND Synchronisation
 -
- Ereignis
 - Startereignis (Einfacher Durchgezogener Kreis)
 - Endereignis (Einfacher kreis mit punkt in der Mitte)
- Interruptable Activity Region
 - alles innerhalb kann abgebrochen werden
 - Exception (gezackter Pfeil) führt zu reagierender Funktion

5. UML Use-Case Diagramme

- beschreiben Anwendungsfälle
- aus Sicht des Anwenders
- gemeinsam mit Anwender

- Ziel ist universelle Verständlichkeit
- für Anforderungserhebung
 - führt zu techn. Systementwurf
 - unterstützt Abnahmetests (erfüllt System Erwartungen)

Ein **Use-Case** ist:

- eine Sequenz von Transaktionen (Einzelschritte)
- Transaktion selbst hat keinen Nutzen
- Aber Use-Case sollte immer Nutzen bringen
- immer Teil eines Systems
- immer in Verbindung mit Akteuren

Akteure (Strichmännchen):

- Ist eine Rolle, keine konkrete Person
- immer außerhalb von Systemgrenzen
- können menschlich sein oder andere Systeme repräsentieren
- benutzen System oder werden von System benutzt
- Linie bedeutet Verbindung/Beziehung mit Use-Case
 - Binär (zwei Beteiligte Use-Case & Akteur)
 - Kardinalitäten möglich (auf Seite des Akteurs => mehrere Akteure führen Use-Case aus)
- auch als Rechteck mit Stereotype “Actor” möglich

Beziehungen zwischen Use-Cases:

- Interaktion mit Akteur
 - es wird der jew. Akteur benötigt um Use-Case auszuführen
- include
 - von A zu B = A inkludiert B
 - wenn A ausgeführt wird muss B ausgeführt werden
- extends
 - von B zu A = B erweitert A
 - A kann von B erweitert werden
 - optionale Erweiterung
 - Erweiterung kann Bedingt werden (Bedingung = Strichlierte Linie mit Notiz ODER Extension Point)
- Generalisierung
 - von B zu A (Spezialisierung)
 - von A zu B (Generalisierung)
 - auch zwischen Akteuren
 - B erbt Funktionalität von A und ergänzt diese
 - abstract = Use-Case kann nicht instanziiert werden
 - Beziehung zu Akteuren wird weitervererbt
 - ODER Beziehung von Akteuren kann mit Generalisierung dargestellt werden

Identifikation von Akteuren

- Gespräch mit Anwender
- wichtige Personen finden/analysieren

Anwendungsfallbeschreibung

- in Textdokument
- Trigger
 - Ereignisse
- Reihenfolge von Transaktionen

Einsatz von Use-Cases

- bei RUP
- bei SCRUM
- werden über jede Iteration immer kürzer & konkreter