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 & Kompnentenentwurf - Systemarchitektur - technische Spezifikation - Schnittstellen - . . .

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

Betriebnahme - 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 änden)
- 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/waranty

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 meherer 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/Implementierugn - **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 & Ellaboration - 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
    - Infrastruktutätigkeiten 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/geschlossen 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 detailierte Informationen erfassbar

Fragebogen

- bei einer größeren Zielgruppe
- eher standardisierte/erprobte Fragebögen verwenden
- Test bei einer kleineren Gruppe (Verständniss & 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