Die Inhalte der SEPM VO aus dem 2018W zum stupiden Auswendiglernen. Alle Angaben ohne Gewähr. Viel Spaß! (Spoiler: Es wird keinen Spaß machen.)

Anmerkung (vielleicht liest es ja mal jemand der eine Änderung bewirken kann (wahrscheinlich nicht, aber die Hoffnung stirbt bekanntlich zuletzt)): Diese Lehrveranstaltung in dieser Form ist (meiner höchstpersönlichen Meinung nach) ein Schlag ins Gesicht eines jeden des Denkens fähigen Menschen und eine gewaltige Verschwendung von Zeit, Energie und Nerven. So etwas hat an einer Universität (und auch sonst nirgends) absolut nichts verloren. Die Vortragenden haben es geschafft, ein ohnehin schon langweiliges Thema so schlecht wie nur irgendwie möglich in eine Lehrveranstaltung zu packen. Eine beachtliche Leistung, angesichts derer es ehrlich gesagt ihrerseits höchst angebracht wäre, einmal ordentlich darüber zu reflektieren, was sie hier eigentlich treiben.
Musste gesagt werden, Danke für die Aufmerksamkeit!

1. Analyse
    1.1. Stakeholder
    Ist jede Person/Institut, die Projekt beeinflussen kann oder davon beeinflusst wird; Typischerweise: Auftraggeber, Geschäftsführung, Legacy Owner, Betreiber, Benutzer, Projektleiter, Entwickler
    1.2. Stakeholdergruppe
    Gruppe von Stakeholdern mit gleichem Interesse; Jede Stakeholdergruppe hat eigene Sicht auf Projekt und eigenen Lingo (= Sprach-Subset?)
    1.3. Woher kommen Anforderungen?
        - Stakeholder
        - Standards/Normen
        - Gesetze
        - Projektumfeld (z.B. Features der Konkurrenz, Unternehmenskultur)
        - Verdeckte Regeln (Glaubensgrundsätze, Werte, Einstellungen, Tabus, Status, Macht)
    1.4. [?] Arten von Zielen die während der Analyse festgestellt werden
        - Strategische Ziele (= langfristige Unternehmensziele, z.B. Besserung Unternehmensimage)
        - Operative Ziele (= mittel- bis kurzfristig, z.B. Einhaltung Projektkosten)
        - Führungsziele (= lang-, mittel- oder kurzfristig, befassen sich mit internen Abläufen und Faktor Mensch, z.B. Steigerung der Mitarbeitermotivation)
    1.5. Vision und Scope
        - Vision: Bleibt normalerweise gleich; = Zweck, Nutzen, Business Case; Was sich (menschlich, ungefähr) vom Projekt erwartet wird
        - Scope: Üblicherweise veränderbar; = Fachlicher Umfang des Projekts; Harte Fakten welche Funktionen die Software haben muss/soll (Use Cases, etc.)
    1.6. Einflussfaktoren Make or Buy
        - Entwicklungskosten
        - Lizenzkosten
        - Anpassungsmöglichkeiten

- Schulungsaufwand
- Support und Wartung
- Herstellerabhängigkeit
- strategische Elemente (z.B. Weiterverkaufsmöglichkeit einer Neuentwicklung)

1.7. Methoden zur Anforderungserhebung
- Workshops (mehrere Stakeholder auf einmal)
- Interviews (ein Stakeholder oder -Vertreter auf einmal)
- Beobachtung (der Anwender des bestehenden Systems; es kann auch nachgefragt werden)
- Mitarbeit (ähnlich wie Beobachtung, aber Analyst benutzt Altsystem selbst mit)

1.8. Stabile und volatile Anforderungen
- Stabile: Ändern sich voraussichtlich nicht/kaum; Können früher und hard-wired bzw. auf simplere Art implementiert werden
- Volatile: Ändern sich voraussichtlich während der Anforderungsanalyse oder des Betriebs des Systems; Sollten mit adaptivem Design bzw. Soft-wired implementiert werden

1.9. Baseline
Basis-Anforderungen, deren Änderung nur formal geht; Echtes Anforderungs-Freeze existiert niemals

1.10. Arten von Anforderungen
- Funktionale
- Nichtfunktionale (Qualität, Entwicklungsprozess, Gesetze, Standards, Normen)
- Domänenanforderungen (Implizite funktionale oder nichtfunktionale Anforderungen, die sich aus der Domäne des Projekts ergeben; Werden oft nicht explizit dokumentiert)

1.11. Strategischer und taktischer Entwurf
- Strategischer Entwurf: Nichtfunktionale Anforderungen
- Taktischer Entwurf: Funktionale Anforderungen

1.12. Qualitätsattribute von Anforderungen
- Komplett/Vollständig: Es darf keine impliziten Annahmen von Stakeholdern geben
- Konsistent: Keine Konflikte/Widersprüche mit/zu anderen Anforderungen
- Eindeutig
- Verständlich
- Notwendig
- Testbar
- Priorisiert
- Identifizierbar: Mittels ID
- Atomar: Nur eine tatsächliche Anforderung pro ID
- Einheitliche und zentrale Dokumentation aller Anforderungen

1.13. Überprüfung von Anforderungen

- Anforderungs-Review: Konflikte erkennen und lösen (z.B. mittels Priorisierung)
- Prototyping: Prüfen, ob Anforderungen umsetzbar, eindeutig und vollständig sind
- Abnahmetestfallerstellung: Testbarkeit, Eindeutigkeit und Vollständigkeit der Anforderungen wird geprüft

1.14. Typische Probleme bei Anforderungen
- Verständlichkeit (Sprache der Domäne wird von Entwicklern nicht verstanden)
- Selbstverständlichkeit (Manche Anforderungen werden nicht explizit genannt)
- Ungenau
- Vermischung von funktionalen und nichtfunktionalen
- Fehlende Abgrenzung unterschiedlicher Anforderungen

1.15. Wie können Anforderungen modelliert werden?
- UML Verhaltensmodelle (z.B. Sequenzdiagramm)
- UML Strukturmodelle (z.B. Klassendiagramm)
- (Logische) Datenmodellierung: Häufig irgendein relationales Modell
- Use Cases
- Geschäftsprozessmodellierung

1.16. Methoden zur Anforderungsanalyse
- Nicht strukturiert: Natürliche Sprache; Häufigste Methode
- Semi-strukturiert: Basieren auf natürlicher Sprache, aber mit Vorlagen, eingeschränktem Vokabular und ergänzenden Properties
- Strukturiert: Folgen definierter Syntax, Notation oder formalen Spezifikation
- Use Cases
- User Stories
- Geschäftsprozessmodellierung, insb. Business Process Modeling Notation (BPMN)

1.17. Inhalt von Use Cases
- ID
- Name
- Beschreibung
- Akteure
- Status (der Umsetzung)
- Vorbedingungen
- Nachbedingung/Ergebnis
- Standardablauf (Normale konkrete Ausprägung dieses Falles)
- Alternative Ablaufschritte
- Änderungsgeschichte

1.18. Vor- und Nachteile von User Stories
+ Kurz und prägnant, somit gut wartbar
+ Fördern Diskussion zwischen Kunde und Entwickler während des gesamten Projekts
+ Reduzieren Risiko von Fehlentwicklungen aus Benutzersicht

- Ohne näher präzisierte Akzeptanz-Testfälle zu ungenau
- Setzen starke Involvierung des Kunden/Benutzers voraus, was oft unrealistisch ist
- Können kaum als Vertragsgrundlage dienen
- Sind personenzentriert

1.19. Anforderungsdokument
- Offizielles Dokument und Vertragsgrundlage
- Kann enthalten: Muss-, Darf nicht-, Soll, Soll nicht- und Kann-Anforderungen

1.20. Anforderungsnachverfolgbarkeit/Traceability
- Zur Quelle: Welcher Stakeholder und wann?
- Zu anderen Anforderungen (wenn gegenseitige Abhängigkeiten existieren)
- Zu abhängigen Artefakten: z.B. Elemente des Systemdesigns oder Testfälle die sich aus dieser Anforderung ergeben

1.21. Integration verschiedener Anforderungsquellen
- Generell können verschiedene Anforderungen auf verschiedene Weisen beschrieben werden (z.B. Formale Anforderungslisten für Sicherheitsaspekte und Use Cases für Bedienung)
- User Stories und Use Cases können sich gegenseitig verfeinern; User Stories sind wesentlich informeller als Use Cases

2. Entwurf

2.1. Ziele des Entwurfs
- Entwurf eines Systems, das die Anforderungen abdeckt
- Beschreibung des Systems auf technischer Ebene
- Festlegung der Architektur des Systems
- Aufteilung des Systems in funktional getrennte Blöcke
- Ausgangspunkt für Implementierung

2.2. Was ist Architektur?
- Besteht aus
  - Strukturen des Systems
  - Zerlegung in Komponenten
  - Schnittstellen der Komponenten
  - Beziehungen zwischen Komponenten
- Ist Hauptträger der Nichtfunktionalen Anforderungen eines Systems

2.3. Entwurfsparadigmen: Grundsätze des Entwurfs
Sind allgemein akzeptierte und abstrakt formulierte Regeln:
- Keep It Small and Simple (KISS)
- Explizit statt Implizit
- Mit Änderungen und Fehlern rechnen
- Qualitätsanforderungen einhalten
- Spezifisch entscheiden, es gibt kein Patentrezept

2.4. Entwurfsparadigmen: Prinzipien des Entwurfs
Sind konkrete Regeln des Softwareengineering:
- Geringe Kopplung (= Abhängigkeit zwischen Komponenten)
- Hohe Kohäsion (= Zusammenhalt innerhalb einzelner Komponenten)

- Separation of Concern (Komponenten haben jeweils unterschiedliche Zuständigkeiten)
- Modularisierung
- Abstraktion, Kapselung, Geheimnisprinzip (Innenleben der Komponenten verbergen, Schnittstellen für den öffentlichen Zugriff)
- Konsistenz
- Don’t Repeat Yourself
- Zyklische Abhängigkeiten vermeiden

2.5. Entwurfsparadigmen
- Objektorientiertes Design allgemein
- Objektorientiertes Design SOLID
- Domain-Driven Design

2.6. Objektorientiertes Design SOLID
- Single Responsibility Principle: ~= Separation of Concern, Eine Komponente soll nur eine Aufgabe haben
- Open-Close Principle: Künftige Erweiterungen sollen ohne Änderung von bestehendem Quellcode möglich sein
- Liskov Substitution Principle: Unterklassen müssen für ihre Oberklassen eingesetzt werden können
- Interface Segregation Principle: Viele kleine Interfaces statt ein großes
- Dependency Inversion Principle: Abhängigkeiten von Abstraktionen, nicht von Implementierungen

2.7. Domain-Driven Design
- Strategisches Design (Clustern im Großen)
  - Unterteilen der Fachdomäne in Bounded Contexts (= thematisch eng zusammenhängende Themenbereiche)
  - Innerhalb eines Bounded Contexts sind Begriffe eindeutig
  - Bounded Context Beispiel für Telefonanbieter: Netzbetrieb, Billing
- Taktisches Design (Strukturieren im Kleinen)
  - Unterteilen eines Bounded Contexts in Internal Building Blocks oder Tactical Patterns

2.8. Architektur im agilen Kontext
- 10-25% der Anforderungen ändern sich pro Jahr
- Architektur muss stabiles Grundgerüst sein, das Umsetzung von neuen und geänderten Anforderungen schnell und effektiv ermöglicht

2.9. Arten von Softwarearchitektur
- 2-Tier Architekturen
- 3-Tier Architekturen
- Service-Orientierte Architekturen
- Microservice Architekturen

2.10. 2-Tier
App + Server

2.11. 3-Tier
App + Server + Database

2.12. Service-Orientierte Architekturen

- Anwendungen durch mehrere Services gelöst
- Services können von mehreren Anwendungen verwendet werden
- Services können andere Services aufrufen
- Paradigmen:
    - Lose Kopplung
    - Abstraktion (wenn möglich)
    - Standardisierung

2.13. Business Process Modelling und SOA

- Sind eng miteinander verknüpft, weil:
- Service-Orientierte Architekturen auf Geschäftsprozessen basieren

2.14. Microservice-Architekturen

+ Erhöhung der Wandelbarkeit und Flexibilität
+ Kürzere Time-to-Market von neuen Features
+ Schnellere Entwicklung weil kleinere Einheiten
+ Mehr Flexibilität bzgl. Technologieauswahl
+ Microservices unabhängiger voneinander
- Organisation muss entsprechend angepasst werden:
    - Teams sollten sich von Anfang bis Ende um ihre Microservices kümmern (“You build it, you run it”)
    - Team 1 : n Microservices
- Service-Discovery
- Dezentralisierte Daten: Konsistenz?
- Viele Laufzeitumgebungen: Aufwand

2.15. Patterns

Stellen allgemeine, abstrakte Lösungsstrategien für wiederkehrende Probleme der Softwareentwicklung dar

2.16. Arten von Patterns

- Architektur Patterns
    - auf Architektur-Ebene
    - Schablonen für konkrete Architekturen
    - Systemweite, strukturelle Eigenschaften einer Anwendung
    - z.B.: Model-View-Controller, Pipes and Filters, Layered Architecture
- Design Patterns
    - auf Klassen-Ebene
    - Wiederholt auftretende Strukturen von miteinander kommunizierenden Komponenten
    - Zum Lösen eines allgemein definierten Problems innerhalb eines bestimmten Kontexts

3. Implementierung, Integration, Testen und Inbetriebnahme

3.1. Ziele der Implementierung

- Umsetzung der Anforderungen
- Umsetzung der Architektur in Software
- Zeitgerechte Lieferung
- Entwicklung wartbarer Software

3.2. Voraussetzungen für die Implementierung

Es muss vorhanden/festgelegt sein:

- Architektur
- Team
- Entwicklungsprozess
- Programmiersprache(n)
- Tools
- Zeitplan

3.3. Unterschied Library und Framework

- Library wird vom Entwickler-Code aufgerufen
- Framework ruft Entwickler-Code auf (“Inversion of Control” / “Hollywood-Prinzip”)

3.4. Version Control System (VCS)

- Zentral: z.B. SVN (Subversion), CVS (Concurrent Versions System)
- Dezentral: z.B. git

3.5. Build Management System (BMS)

- Werkzeug zum Bauen einer Applikation oder Teilen/Modulen davon
- z.B. Make, Maven, Ant, Gradle

3.6. Wartungsfreundliche Software

Zeichnet sich aus durch:

- Niedrige Kopplung
- Hohe Kohäsion
- Schnittstellen
- Information hiding (Komponente soll von anderer Komponente so wenig wie möglich wissen (können))

3.7. Design Patterns

- Creational Patterns
  - Abstract Factory
  - Builder
  - Singleton
  - Prototype
- Structural Patterns
  - Adapter: “Übersetzt” zwischen Interfaces; z.B.: Wrappt Objekt und führt Originalfunktionen so aus, dass es anderem, ursprünglich nicht implementierten Interface gerecht wird
  - Bridge
  - Decorator
  - Facade
  - Proxy
- Behavioral Patterns
  - Command
  - Iterator
  - Observer
  - Strategy

3.8. Test-Driven Development (TDD)

- Es werden zuerst die Tests geschrieben und erst danach die eigentliche Umsetzung

- 2 wichtige Regeln:
    - Neuer Code darf nur geschrieben werden, wenn ein (neuer) Test fehlschlägt
    - Alle Redundanzen im Code müssen beseitigt werden

3.9. Grundsätze des Softwaretests

nach ISTQB (International Software Testing Qualifications Board):

- Testen zeigt die Anwesenheit von Fehlern
- Vollständiges Testen ist unmöglich
- Mit dem Testen früh beginnen
- Häufung von Fehlern (was auch immer das heißen soll)
- Tests müssen überarbeitet und erweitert werden
- Testen ist abhängig vom Umfeld (aha, danke)
- Erfolgreiche Testausführung bedeutet nicht dass das System fehlerfrei ist

3.10. Ziele des Softwaretestens

- “Verifikation”: Prüfen des Systems gegen Spezifikation (“Erstellen wir das Produkt richtig?”)
- “Validierung”: Prüfen des Systems gegen (Kunden-)Anforderungen (“Erstellen wir das richtige Produkt?”)

3.11. Methodik des Softwaretestens

- Die Testfälle identifizieren, mit denen am besten geprüft werden kann, ob das System korrekt funktioniert
- Möglichst hohe funktionale und nichtfunktionale Abdeckung

3.12. Test- und Integrationsstufen

- Pro Projektphase genau eine Testphase
    - Architekturdesign: Systemintegrationstest
    - Technisches Design: Komponentenintegrationstest
    - Modul-/Klassendesign: Komponenten-/Unittest

  Projektphasen laufen der Reihe nach ab, dann kommt die Implementierung und danach beginnen die jeweiligen Teststufen in der umgekehrten Reihenfolge, d.h. von Komponenten-/Unittests zu Systemintegrationstest

3.13. Komponenten-/Unittests

Testen ein einzelnes Modul, Klasse, etc. z.B. auf

- Robustheit
- Ihre Struktur (z.B. Zweigüberdeckung)
- Ihren Umgang mit Ressourcen

Benötigen Isolation der Komponente vom restlichen System, was wiederum saubere Schnittstellen erfordert

3.14. Integrationstests

Testen Schnittstellen und Interaktion zwischen Komponenten, verschiedene Strategien:

- Big-Bang-Integration
- Horizontale Integration: Integration Schicht für Schicht
    - Top-Down-Integration
    - Bottom-Up-Integration

- Vertikale Integration Integration pro Funktionalität/Use Case

3.15. Big-Bang-Integrationstest

Alles, also das gesamte System, wird auf einmal getestet

+ Keine Stubs
+ Keine Testtreiber
- Erst spät möglich (was wiederum ein Risiko darstellt)
- Fehleridentifizierung und -lokalisierung ist schwierig

3.16. Horizontale Integration

Iterative, schichtorientierte Integration

- Top-Down: Höhere Anwendungsschichten werden zuerst umgesetzt, die unteren werden simuliert (in sog. Stubs)
  + Externe Schnittstellen früh verfügbar
  - Hoher Simulationsaufwand (wegen den Stubs)
- Bottom-Up: Untere Anwendungsschichten werden zuerst umgesetzt, es existieren mehrere Testtreiber um von jeder Schicht “wegstarten” zu können
  + Keine Stubs notwenig (dafür mehr Testtreiber, aber trotzdem weniger Aufwand)
  - Externe Schnittstellen erst spät verfügbar

3.17. Vertikale Integration

Iterative, funktionsorientierte Integration

- Pro Use Case werden immer die Komponenten, die dafür notwendig sind, gemeinsam getestet

+ Funktionalität früh verfügbar und abnehmbar
+ Lauffähiges, testbares Teilsystem nach jeder Iteration
+ Gut geeignet für iterative Entwicklungsprozesse
- Alle Schichten müssen die jeweilige Funktionalität unterstützen
- Synchronisation in der Entwicklung notwendig

3.18. Systemtest

Testet das gesamte System in Umgebung äquivalent zur Produktivumgebung

3.19. Akzeptanztest

- Meist vom Auftraggeber durchgeführt
- Prüft Umsetzung der vereinbarten Leistungen
- Arten:
  - Anwender-Akzeptanztest: Benutzbarkeit für Anwender
  - Betrieblicher: Backupmöglichkeit, Wiederherstellbarkeit, etc.
  - Regulatorischer: Gesetzes-/Standardkonformität
  - Feldtest: ?

3.20. Funktionale Softwaretests

- Ziel: Funktionale Verifikation des Systems
- Methodik: Hohe Testabdeckung mit Testfällen, die mit hoher Wahrscheinlichkeit Fehler finden
- Ansätze:
  - Strukturelle Abdeckung (“White Box Test”)
  - Funktionale Abdeckung (“Black Box Test”)

3.21. Strukturelle Abdeckung (“White Box Test”)

- Testfälle aus innerer Struktur des Systems ableiten
- Analyse des Kontrollflussgraphs:
    - Anweisungsüberdeckung (= alle möglichen Anweisungen müssen einmal ausgeführt werden)
    - Bedingungsüberdeckung (= alle Auswertungen aller elementaren Bedingungen müssen einmal true und einmal false ausfallen)
    - Pfadüberdeckung (= Programm muss alle möglichen Pfade nehmen; vollständige Pfadabdeckung bei Schleifen unmöglich bzw. praktisch unmöglich)

3.22. Funktionale Abdeckung (“Black Box Test”)

Mittels:

- Äquivalenzklassenanalyse (s.u.)
- Zustandsbasierte Testmethoden
    - Ableiten der Testfälle aus Zustandsautomat (z.B. UML-Diagramm)
    - Alle Zustände, Zustandsübergänge und Auslöser von Zustandsübergängen durchführen
- Klassifikationsbaummethode (s.u.)
- Informelle Testmethoden

3.23. Äquivalenzklassenanalyse

- Einteilung der möglichen Eingabewerte in Äquivalenzklassen mit gleichem erwartbaren Systemverhalten
- Tests werden dann nur mit einem Input pro Äquivalenzklasse durchgeführt

Äquivalenzklassenanalyse mit Grenzwertanalyse:

- Äquivalenzklassen:
    - $x < 0$
    - $0 <= x <= 99$
    - $99 < x$
- Testfälle:
    - -1
    - 0
    - 1
    - 99
    - 100

3.24. Klassifikationsbaummethode

- Testfälle bestehen aus Kombination von (Programmcode-)Klassen
- Nach Möglichkeit alle möglichen Kombinationen durchtesten?

3.25. Software Life Cycle

- Analyse
- Entwurf
- Implementierung
- Test
- Inbetriebnahme
- Wartung (kontinuierlich)

- → Goto Analyse (für Weiterentwicklung)

3.26. Anwenderakzeptanz

Kann erreicht werden durch

- Anwendergerechte Dokumentation (Benutzerhandbuch, Onlinehilfe)
- Schulungen
- Stückweises Rollout (Kein “Big Bang”-Rollout, Testbetrieb bereits während der Entwicklung, Parallelbetrieb des alten und des neuen Systems)

3.27. Inbetriebnahme

Muss geplant werden!

- Eigene Maschine(n)?
- Datenbank
- Integration von Fremdsystemen
- Verantwortlichkeiten (Installation der Software, Einrichten der Datenbank, Benutzerkonten)
- Mit Dokumentation: Release-Checkliste, Release-Drehbuch

Best Practices

- Zusammenarbeit Entwickler und Betrieb
- Verantwortlichkeiten festlegen
- Frühzeitiges Testrollout

Anti-Patterns

- Manuelle Installation der Software
- Installation erst nach Abschluss der Entwicklung
- Manuelle Konfiguration
- Änderungen direkt in der Betriebsumgebung (“Hot Fixes”)

3.28. Definition Softwarewartung

- Änderungen einer Software
- Nach der Lieferung
- Zur Fehlerbehebung, Verbesserung oder Anpassung
- Die die Funktionalität nicht ändern

3.29. Wartungstypen

- Korrektiv (Aufgetretene Fehler ausbessern)
- Präventiv ((Noch) nicht aufgetretene Fehler ausbessern)
- Adaptiv (Anpassung an geänderte Umgebung)
- Perfektionierend (Verbesserung, z.B. Performance)

3.30. Maßnahmen der Wartung

- Reengineering: Neuentwicklung eines funktionierenden Systemteils
- Refaktorisierung: Umstrukturierung bei gleichbleibender Funktionalität
- (Eliminieren von Anti-Patterns: Entfernen von schlechten Softwareteilen (== Refaktorisierung?))

4. Vorgehensmodelle, Qualitätssicherung, Management

4.1. Traditionelle Vorgehensmodelle

- Fundamentales V-Modell nach Böhm 1992
- V-Modell 97
- Spiralmodell nach Böhm 1988
- Rational Unified Process

4.2. Traditionell: Fundamentales V-Modell nach Böhm 1992
Wie Teststufen: Implementierungsphasen links und dann in die andere Richtung die entsprechenden Testphasen rechts

4.3. Traditionell: V-Modell 97
Funktionen werden priorisiert und einer Ausbaustufe zugeordnet; Jede Ausbaustufe wird dem Kunden nach Fertigstellung zur Verfügung gestellt

4.4. Traditionell: Spiralmodell nach Böhm 1988
Folgende Phasen werden wiederholt bis fertig:
- Definition von Zielen
- Risikoanalyse
- Durchführen der Arbeitsschritte
- Planen der nächsten Phase(n)

4.5. Traditionell: Rational Unified Process
Prinzipien:
- Software iterativ entwickeln
- Komponentenbasierte Architekturen verwenden
- Software visuell modellieren (UML)
- Softwarequalität kontinuierlich prüfen
- Änderungen kontrolliert in die Software einbringen

Für jedes Projekt ist selektive Adaption des RUP vorzunehmen (vom sogenannten Process Engineer)

4.6. Agile Vorgehensmodelle
- Scrum
- XP
- XP@Scrum
- Kanban

4.7. Agile Manifesto (2001)
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

4.8. Agil: Scrum
Elemente:
- Sprint
- Daily Scrum
- Product Backlog
- Increment: ~Ergebnis des letzten Sprints (Software)
- Burndown Chart

Rollen:
- Product Owner: Erstellt Product Backlog, hält das Team von Unterbrechungen frei
- Scrum Team
- Scrum Master: Verwaltet das Team und schirmt es von äußeren Einflüssen ab

4.9. Agil: Extreme Programming (XP)
- Kompakt

- Für kleine bis mittelgroße Teams
- Für ungenaue oder sich schnell ändernde Anforderungen

Prinzipien:

- Ständige Kommunikation
- Einfachheit: Das simpelste Design das die Aufgabe löst
- Feedback
- Mut: Zur Offenheit und zu Änderungen
- Respekt

4.10. Agil: XP@Scrum

- Scrum als Basis
- Zusätzlich Elemente von XP, z.B.:
  - Pair Programming
  - Code Reviews
  - Continuous Integration

4.11. Agil: Kanban

Keine vorgeschriebenen Rollen, Meetings, etc., außer:

- Kontinuierlicher “Pull” Workflow
- Kanbanboard
- Work in Progress Limit

4.12. Starplot/Entscheidungsfaktoren Traditionell/Agil nach Böhm und Thurner

- Teamqualifikation: Agile Methoden erfordern qualifizierteres Team
- Änderungsrate der Anforderungen: Je höher, desto einfacher muss Änderung möglich sein
- Kultur: Je nach Umfeld können agile Methoden eingesetzt werden
- Teamgröße
- Kritikalität: Je wichtiger das System ist (vor allem wenn Menschenleben davon abhängen), desto mehr muss das System geprüft werden

4.13. Methoden der Softwarequalitätssicherung

- Statische analytische Quaitätssicherung
  - Code-Metriken
  - Review/Walkthrough/Inspection
- Dynamische analytische Qualitätssicherung
  - Testen
  - Dynamische Analyse (aha, danke)
- Organisatorische Qualitätssicherung
  - Templates
  - Checklisten
  - Wissensmanagement
  - Qualitätsmanagementstandards (no shit)

4.14. Statische Qualitätssicherung

- Prüfung ohne dass das Testobjekt ausgeführt wird
- Bereits sehr früh möglich
- z.B. Entwicklungsrichtlinien, Reviews

4.15. Statische Qualitätssicherung: Softwaremetriken

- Prozessmetriken: z.B. Wie schnell werden Fehler entdeckt? Welche Aufwände verursachen Fehler? etc.
- Produktmetriken: z.B. Lines of Code, Modularität, Kopplung, Testergebnisse, Testabdeckung, etc.

4.16. Statische Qualitätssicherung: Reviews

Arten:

- Stellungnahme: Schriftlich, kein Meeting
- Walkthrough: Weniger formell, Meeting(?)
- Inspektion bzw. Technisches Review: Beurteilung durch ein qualifiziertes Team; Fixer Prozess und Techniken; Formellste und wirkungsvollste Review-Art
- Round-Robin-Review: Gutachter erhalten abwechselnd die Möglichkeit, die Führungsrolle zu übernehmen
- Peer-Review: Kein fixer Prozess oder Techniken

4.17. Dynamische Qualitätssicherung

- Prüfung während der Laufzeit
- System wird mit Testdaten ausgeführt
- Hierunter fallen alle möglichen Arten von Tests (Unittests, Integrationstests, Systemtests und mehr falls es noch welche gibt)

4.18. Organisatorische Qualitätssicherung

Ziel: Vermeidung/Verringerung der Fehlerrate

- Wissensmanagement
- Konfigurationsmanagement
- Dokumentenvorlagen
- Checklisten

4.19. Qualitätsfaktoren

Ermöglichen Aussagen über die Qualität von Software

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

4.20. Qualitätskosten

Sind Summe von

- Prüfkosten
- Fehlerkosten

4.21. Gesamtkosten

Sind Summe von

- Herstellungskosten
- Qualitätskosten

4.22. Qualitätsmanagementstandards

- Zertifizierungsstandards
- Assessmentstandards (Bewertungsstandards)
    - CMM und CMMI (Capability Maturity Model Integration)

        - Beurteilen eine Organisation im Hinblick auf ihre Fähigkeit, Software unter bestimmten Rahmenbedingungen zu erstellen

5. Projektmanagement, Risikomanagement
    5.1. Magisches Dreieck
    Dreieck mit Ecken:
    - Qualität
    - Kosten
    - Zeit

    5.2. Teufelsquadrat
    Quadrat mit Ecken:
    - Qualität
    - Kosten
    - Zeit
    - Umfang

    Inneres Viereck dessen Flächeninhalt die erforderliche Produktivität repräsentiert. Je näher die Ecken des inneren Vierecks an den jeweiligen Ecken des äußeren Quadrats, desto mehr kommt der jeweilige Faktor ins Spiel

    5.3. Projektorganisationen:
    - Reine
    - Matrix
    - Einfluss

    5.4. Projektorganisation: Reine Projektorganisation
    - Projektmitarbeiter sind nur dem Projekt zugeteilt
    - Projektleiter hat volle Weisungsbefugnis
    + Verantwortung ist eindeutig geregelt
    + Starke Identifikation der Mitarbeiter mit dem Projekt
    + Hohe Motivation
    - Schwierige Akquirierung der Mitarbeiter
    - Wiedereingliederung der Mitarbeiter am Ende schwierig

    5.5. Projektorganisation: Matrix-Projektorganisation
    - Projekt teilt sich Mitarbeiter mit restlicher Organisation
    - Projektleiter hat nicht volle Weisungsbefugnis
    + Sicherheitsgefühl der Mitarbeiter
    + Flexibler Personaleinsatz
    - Mitarbeiter haben jeweils 2 Zuständigkeiten
    - Konfliktpotenzial Projekt/Rest
    - Hoher Koordinationsaufwand

    5.6. Projektorganisation: Einfluss-Projektorganisation
    - Projekt hat keine fixen Mitarbeiter
    - Projektleiter hat kaum Weisungsbefugnis
    + Kaum organisatorische Veränderung notwendig
    + Hohe personelle Flexibilität
    - Keine klare Verantwortlichkeit
    - Fehlende Autorität des Projektleiters
    - Hohes Konfliktpotenzial

- Fehlender Teamgeist

5.7. Projektplanung: Vorgangsweise
- Projektstrukturplan
- Arbeitspakete
- Aufwandsschätzung
- Ablauf- und Terminplanung
- Ressourcenplanung
- Kostenplanung
- Optimierung

5.8. Projektstrukturplan: Vorgangsweise
- Top-Down
  - Stufenweise Zerlegung der Gesamtaufgabe
- Bottom-Up:
  - Sammlung kleiner Aufgaben
  - Gruppierung nach bestimmtem Kriterium
  - Ergänzung um fehlende Aufgaben

5.9. Arbeitspakete
- Kleinste Einheiten des PSP
- Umfang kann sehr unterschiedlich sein
- Wichtig:
  - Konkretes Ergebnis
  - Kalkulierbarkeit
  - Keine Überschneidung mit anderen Paketen
  - Klare Schnittstellen
  - Eindeutige Verantwortlichkeit

5.10. Aufwand vs. Durchlaufzeit
- Aufwand = Insgesamte Arbeitszeit
- Durchlaufzeit = Zeit zwischen Start- und Enddatum

5.11. Vorgangsbeziehungen

Sind Abhängigkeiten der Start-/Endzeitpunkte von Vorgängen untereinander
- Ende-Anfangs-Beziehung (EA): Sequenzielle Abarbeitung: Ende von Vorgang 1 ist Voraussetzung für Start von Vorgang 2
- Anfangs-Anfangs-Beziehung (AA): Parallele Abarbeitung
- Ende-Ende-Beziehung (EE): Parallele Abarbeitung
- Anfangs-Ende-Beziehung (AE): Parallele oder sequenzielle Abarbeitung: Start von Vorgang 1 ist Voraussetzung für Ende von Vorgang 2

5.12. Berechnung der Termine
- Vorwärtskalkulation: Anfangstermin ist festgelegt
- Rückwärtskalkulation: Endtermin ist festgelegt

5.13. Meilensteine

Wichtig, da ohne Meilensteine stärkere Kursabweichungen auftreten werden

5.14. Pufferzeiten von Vorgängen
- Gesamtpuffer: Zeitspanne, um die sich ein Vorgang verzögern darf, ohne dass das Projektende verzögert wird

- Freier Puffer: Zeitspanne, um die sich ein Vorgang verzögern darf, ohne dass andere Vorgänge verzögert werden

5.15. Kritischer Vorgang und kritischer Pfad
- Kritischer Vorgang: = Vorgang ohne Pufferzeit
- Kritischer Pfad: = Pfad, der nur kritische Vorgänge enthält

5.16. Projektcontrolling: Projektüberwachung
Ziel: Rechtzeitiges Erkennen von Abweichungen gegenüber Plandaten
- Durch Kontrollen: Reaktion erst nach eingetretener Abweichung möglich
- Durch Trendanalysen: Aktion schon vor tatsächlichem Eintreten der Abweichung möglich

5.17. Projektcontrolling: Projektsteuerung
Ziel: Projekt auf Kurs halten
- Korrektive Maßnahmen: Angleichung Ist an Soll
  - Mitarbeiter motivieren
  - Kostenkontrolle verstärken
- Reaktive Maßnahmen: Angleichung Soll an Ist
  - Verschiebung der Termine
  - Erhöhung des Budgets
  - Erweiterung des Projektteams
  - Reduktion des Funktionsumfangs

5.18. Risikofaktor
= Risikowahrscheinlichkeit x Schadenshöhe

5.19. Risikomanagement
- Risikoidentifikation: Auffinden von konkret formulierten Risiken
- Risikoanalyse: Bewertung der gefundenen Risiken auf Eintrittswahrscheinlichkeit und Schadensausmaß
- Risikobehandlung/-kontrolle: Erstellung eines Maßnahmenplans zur Behandlung von Risiken

5.20. Teambildungsprozess
- Forming
- Storming
- Norming
- Performing

5.21. Merkmale erfolgreicher Teams
- Zusammenhalt
- Fachliche und soziale Kompetenz
- Unterstützung und Anerkennung von Außen
- Akzeptierte, klare Rollen- und Aufgabenverteilung
- Teamleiter nicht autoritär oder dominant
- Jeder Beitrag wird aufgenommen und gewürdigt
- Konflikte werden offen angesprochen und geklärt
- Erfolgsorientiertes Motivationssystem
- Personen mit Entscheidungskompetenz sind im Team

5.22. Führungsstile
- Autokratisch

- Kooperativ/”teamorientiert”
- Demokratisch
- Situativ
- Authentisch

5.23. Führungsstil: Autokratisch
- Projektleiter bestimmt alles
- Genaue Einzelanweisungen und Kontrolle
- Gefühl der Sicherheit bei unerfahrenen Mitarbeitern
- In Krisen zielführend, da Entscheidungen rasch umgesetzt werden
- Langfristig werden Mitarbeiter gebunden, die Übernahme von Verantwortung scheuen

5.24. Führungsstil: Kooperativ, aka “teamortientierte Führung”
- Projektleiter delegiert Aufgaben, Befugnisse und Verantwortung
- Transparenz bei Entscheidungen und Maßnahmen
- Starke Betonung der Eigenverantwortlichkeit
- Projektleiter muss über Kenntnisse geeigneter Techniken der Intervention verfügen

5.25. Führungsstil: Demokratisch
- Entscheidungen werden durch Gruppendiskussion und -beschluss getroffen
- Projektleiter schlägt Entscheidungen vor, entscheidet aber nicht selbst
- Hohe Gruppenkohäsion und Interaktion
- Starke Motivation und Gruppenmoral
- Qualität und Originalität der Leistung sind hoch
- Geschwindigkeit und Effizienz von Entscheidungen sind oft unbefriedigend

5.26. Führungsstil: Situativ
- Situationselastischer Einsatz verschiedener Führungsstile
- Grundhypothese ist, dass wechselnder Führungsstil sinnvoller ist als immer gleichbleibender Führungsstil
- Besonders für sehr heterogene Projektstrukturen geeignet

5.27. Führungsstil: Authentisch
- Individuelle Persönlichkeit steht im Vordergrund
- Grundwerte:
  - Akzeptanz
  - Respekt
  - Vertrauen
  - Toleranz
  - Offenheit für Neues

5.28. Faktor Mensch

Wichtig:
- Projektmitglieder müssen glücklich sein und Spaß an der Arbeit haben
- Gute Vertrauensbasis
- Freies Arbeiten
- Projektleiter fungiert als Coach, Moderator und Steuermensch