

Software Engineering

Mark Keinhörster

FOM
Hochschule für Ökonomie und Management

3. Juni 2017

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

1 Voraussetzungen

2 Einführung

3 Vorgehensmodelle

4 Projektmanagement

5 Requirements Engineering

6 Exkurs

Das Skript wurde in Anlehnung an die Vorlesungen

- Allgemeine Methoden des Software Engineering, Prof. Dr. Dirk Wiesmann
- Spezielle Methoden des Software Engineering, Prof. Dr. Wolf Ritschel

erstellt und enthält Auszüge aus den jeweiligen Skripten der Dozenten.

Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale Prozessmodelle
- Agile Prozessmodelle

Projektmanagement

- Risikomanagement
- Qualitätssicherung
- Messen/Bewerten

Requirements Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter

Voraussetzungen

Was möchten Sie gerne behandeln?

-
-
-
-
-
-

Was sollten Sie am Ende können?

- Software Engineering als Teildisziplin der Informatik kennen
- Grundpfeiler des Software Engineering kennen
- Vorgehensmodelle der Softwareentwicklung beschreiben und abgrenzen
- Softwarequalität messen und bewerten
- Weiterführende Konzepte verstehen und anwenden

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Einführung

Aufgabe: Rekursives Take

Implementieren Sie die Methode “take(int n)”, die die ersten n Elemente eines Übergebenen Arrays vom Typ int als neues Array zurückgibt.



Die Realität

- Anforderungen mehrere 100 Seiten lang
- Anforderungen unklar, widersprüchlich, flexibel
- International verteilte Teams
- Mehrere tausend Nutzer in 5 Ländern
- Unterstützung von Chrome, Firefox, IE 6
- 6 Monate Projektlaufzeit, 500.000 LOC



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Standish Group (<http://www.standishgroup.com>) veröffentlicht jährlich "Chaos Report".

Chaos Report 2015

- 19% der betrachteten IT-Projekte scheitern
- 52% der betrachteten IT-Projekte drohen zu scheitern
- 29% der betrachteten IT-Projekte sind erfolgreich

On June 4, 1996, on its maiden flight, the Ariane-5 was launched and performed perfectly for approximately 40 seconds. Then it began to veer off course. At the direction of the Ariane ground controller, the rocket was destroyed by remote control. . . total cost of the disaster was 500 million dollar.

- Flugbahn wird durch “Inertial Reference System (SRI)” gemessen, dessen Software teilweise von Ariane-4 übernommen wurde.
- Andere Flugbahndaten erzeugten Überlauf bei Konvertierung von 64-Bit Floating Point in 16-Bit Integer und verursachten Fehlfunktion des SRI-Systems.

- Heartbeat hält TLS-Verbindung am Leben
- Eine Seite schickt beliebig langen Payload, Gegenseite schickt die gleichen Daten wieder zurück
- Indikator für aufrechte Verbindung

Softwarekatastrophe Heartbleed

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale Prozessmodelle
- Agile Prozessmodelle

Projektmanagement

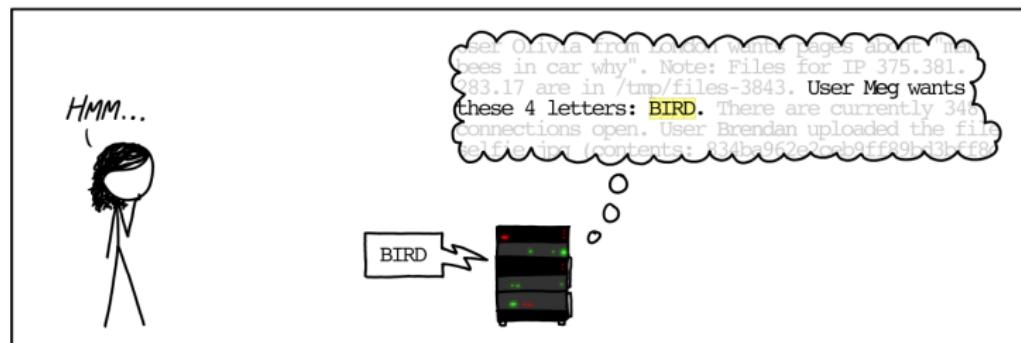
- Risikomanagement
- Qualitäts sicherung
- Messen/Bewerten

Requirements
Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter



- Prüfung ob Payload der angegebenen Länge entspricht fehlte
- War der Payload kürzer als angegeben wurden Daten aus den darauffolgenden Speicherbereichen kopiert
- Da OpenSSL eine eigenen Speicherverwaltung implementiert waren diese Daten auch aus dem OpenSSL Kontext

Softwarekatastrophe Heartbleed

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

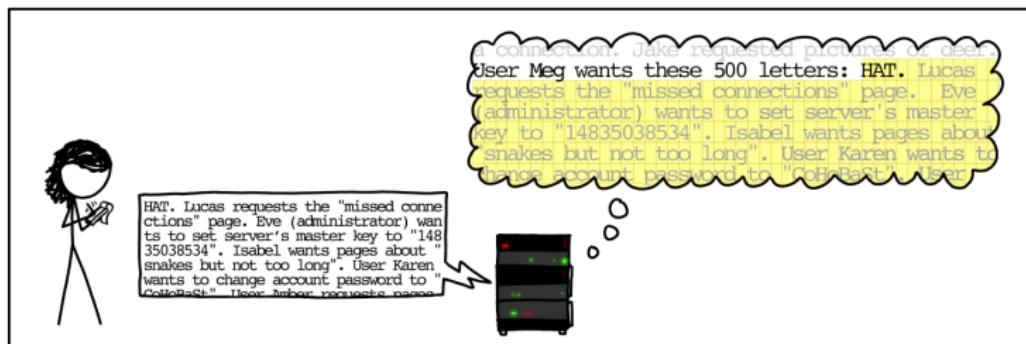
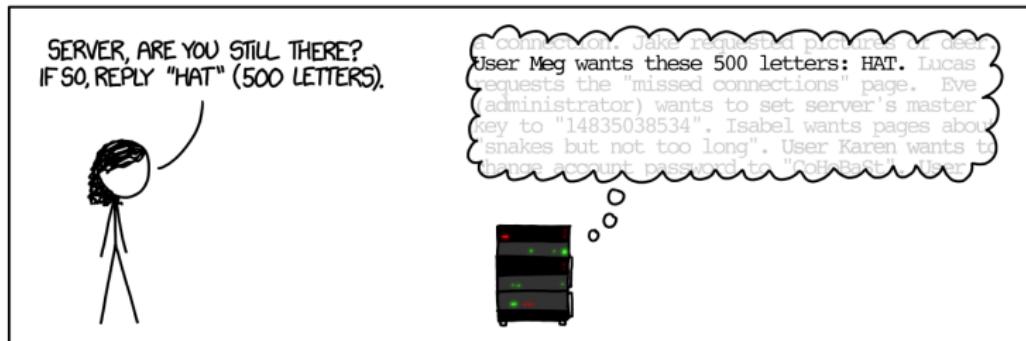
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



SW-Entwicklung in den 40er und 50er Jahren

- Teure Hardware
- Low-Level Programmierung (Assembler, fast kein OS)
- Von Experten bedient (Entwickler = Nutzer)
- numerisch-naturwissenschaftliche Probleme
- Codierung bekannter, mathematisch fundierter Algorithmen
- Viele Daten, einfache Algorithmen
- Häufig Batch-Systeme
- Fokus auf Effizienz
- Häufig "Wegwerf-Software"

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

SW-Entwicklung in den 60er Jahren bis heute

- Preiswerte Hardware mit viel Leistung
- Embedded Hardware die günstig ist und häufig eingesetzt wird
- Nicht-Informatiker nutzen die Software
- Vielzahl von Anwendungsbereichen
- Kritische Anwendungsbereiche wie Finanzsektor etc.
- Systeme sind komplex und interaktiv
- Software teurer als Hardware
- Lange Lebensdauer

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Softwarekrise

- Programme werden immer komplexer
- Passende Programmiersprachen, Methoden, Werkzeuge fehlen

Folgen

- Kosten für Software steigen
- Softwareprojekte scheitern

Lösungsansatz

SW-Entwicklung als Ingenieurstätigkeit mit definiertem Vorgehen statt künstlerischer Tätigkeit

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Dijkstra (The Humble Programmer)

Als es noch keine Rechner gab, war auch das Programmieren noch kein Problem, als es ein paar leistungsschwache Rechner gab, war das Programmieren ein kleines Problem und nun, wo wir gigantische Rechner haben, ist das Programmieren zu einem gigantischen Problem geworden. In diesem Sinne hat die elektronische Industrie kein einziges Problem gelöst, sondern nur neue geschaffen. Sie hat das Problem geschaffen, ihre Produkte zu nutzen.

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

IEEE Definition

Software ist eine Sammlung von Computerprogrammen, Prozeduren, Regeln, zugehöriger Dokumentation und Daten

- Programme sind eine Teilmenge von Software
- SW beeinhaltet Dokumente die verschiedene Abstraktionsschichten für verschiedene Zielgruppen beschreiben

- Kommunikationsprobleme mit Anwender
- SW ist immateriell
- SW ist leicht modifizierbar, Behebung von Fehlern wird unterschätzt
- SW ist nur beobachtbar
- Anforderungen ändern sich regelmäßig
- SW altert über Umgebung, ohne zu verschleißen, dass führt zu immer neuen Erweiterungen und wachsender Komplexität
- Verhalten für Software lässt sich nur schwer beweisen
- ...

- Auslöser für Begriff “Software Engineering” war Softwarekrise von 1968
- Begriff “Software Engineering” wurde 1967 von F.L. Bauer (ehemaliger Prof. in München) im Rahmen einer “Study Group on Computer Science” der NATO geprägt.
- Software wurde erstmals als Industrieprodukt bezeichnet

Definition IEEE

Software Engineering ist der systematische Ansatz für

- die Entwicklung,
- den Betrieb
- sowie die Wartung

von Software.

Definition Lehrbuch Software-Technick (Balzert)

Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notationen und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Software-Systemen. Zielorientiert bedeutet die Berücksichtigung z.B. von Kosten, Zeit, Qualität.

Effiziente Entwicklung von messbar qualitativ hochwertiger Software

- Korrektheit und Zuverlässigkeit
- Robustheit
- Effizienz
- Benutzerfreundlichkeit
- Wartbarkeit und Wiederverwendbarkeit

Qualitätsfaktoren

- Extern (für den Benutzer sichtbar)
- Intern (nur für den Entwickler sichtbar)

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Der systematische Ansatz im Software Engineering wird auch als Entwicklungsprozess bezeichnet. Er beeinhaltet eine Reihe von Aktivitäten die zur Entwicklung von Software führen.

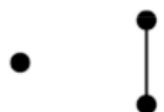
- Spezifikation
- Entwicklung
 - Entwurf
 - Implementierung
- Validierung
- Evolution
 - Weiterentwicklung
 - Betrieb

- Wenige LOC
- SW für die eigene Verwendung
- Produkt spezifiziert sich selbst
- Lösung wird direkt entwickelt
- Validierung und Korrekturen am Endprodukt
- 1 Entwickler
- Komplexität gering
- Software besteht aus wenigen Komponenten
- Wenig bis keine Dokumentation nötig
- Keine Planung und Projektstruktur nötig

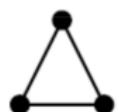
- Viele LOC
- SW für die Verwendung durch Dritte
- Klares Ziel, genaue Spezifikation erforderlich
- Lösung wird in Phasen entwickelt
- Tests in jeder Phase sind unerlässlich
- Produkt wird im Team entwickelt
- Hohe Komplexität macht Strukturierung der SW erforderlich
- Software besteht aus vielen Komponenten
- Dokumentation für den wirtschaftlichen Betrieb der SW erforderlich
- Projektstruktur zwingend erforderlich

Anzahl beteiligte Personen

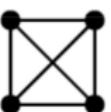
1 2



3



4



5



6



Anzahl Kommunikationspfade

0 1 3



Quantensprung:
Kommunikation
wird erforderlich



Quantensprung: Zahl der
Kommunikationspfade über-
steigt Zahl der Personen

6

10

15

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

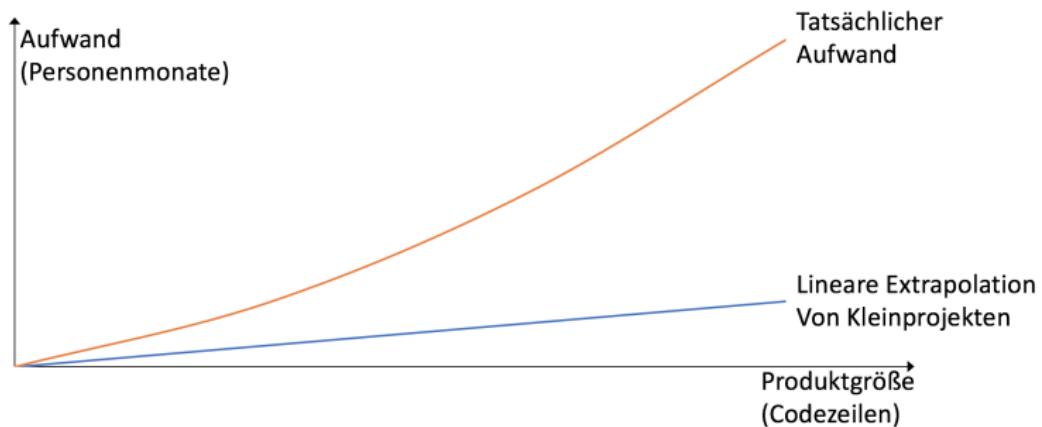
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Wachstum des Aufwands



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

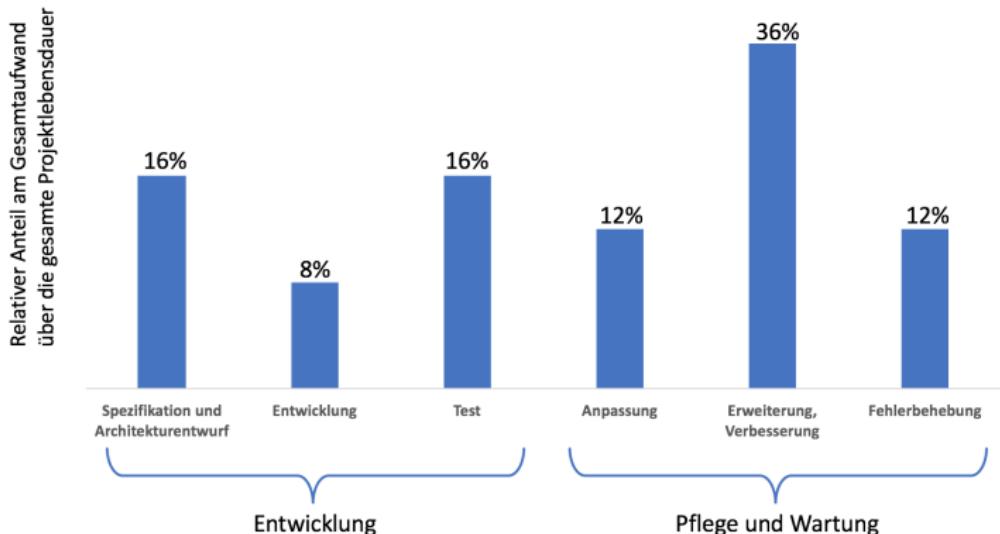
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Aufwände aufgeschlüsselt



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Eine Person braucht zum Bau einer 2m langen Brücke 0,5 Tage. Wie lange brauchen 100 Leute für den Bau einer 2km langen Brücke?

- 1 Interpolieren Sie den Aufwand linear
- 2 Warum ist die Berechnung aus Punkt 1 eine Milchmädchenrechnung?

Eine Person braucht zum Bau einer 2m langen Brücke 0,5 Tage. Wie lange brauchen 100 Leute für den Bau einer 2km langen Brücke?

- 1 Aufwand = $(2000\text{m} / 2\text{m} * 0.5\text{PT}) / 100 \text{ Personen} = 5 \text{ Tage}$
- 2 Mehr Kommunikation, Projekt deutlich Komplexer, Ressourcenbeschaffung, Logistik ...

Übung 1.2

Eine Kundenbetreuerin im Firmenkundengeschäft einer Bank hat auf Grundlage eines Tabellenkalkulationsprogramms eine kleine persönliche Anwendung geschrieben, die sie bei der Überprüfung der Kredite der von ihr betreuten Firmen unterstützt. Die notwendigen Daten gibt sie jeweils von Hand ein. Der Abteilungsleiter sieht diese Anwendung zufällig, ist davon angetan und beschließt, sie allen Kundenbetreuerinnen und -betreuer zur Verfügung stellen. Die notwendigen Daten sollen jetzt automatisch aus den Datenbanken der Bank übernommen werden. Die Kundenbetreuerin gibt an, für die Entwicklung ihrer Anwendung insgesamt etwa vier Arbeitstage aufgewendet zu haben. Der Abteilungsleiter veranschlagt daher für die Übernahme und die gewünschten Änderungen einen Aufwand von einer Arbeitswoche. Als die geänderte Anwendung endlich zur Zufriedenheit aller Beteiligten läuft, sind jedoch rund acht Arbeitswochen Aufwand investiert. Der Abteilungsleiter erzählt die Geschichte einem befreundeten Berater als Beispiel, dass Informatikprojekte nie ihre Termine einhalten. Darauf meint der Berater trocken, der investierte Aufwand sei völlig realistisch und normal. Begründen Sie warum.

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Übung 1.2 - Lösung

- Die Kundenbetreuerin hat das Fachkonzept ihrer Tabellenkalkulationsanwendung vermutlich schon vor den angegebenen vier Tagen Bearbeitungszeit im Kopf gehabt. Die Fremdentwickler müssen dieses zumindest erst nachvollziehen.
- Die neue Anwendung ist durch die Datenbankanbindung mit den entsprechenden Schnittstellen und Zugriffsrechteproblematiken deutlich komplexer.
- Der Kommunikationsaufwand schon allein von Kundenseite (viele Berater = viele unterschiedliche Meinungen) ist erheblich
- Die neu entstandene "professionelle" Anwendung hat einen erheblich höheren Aufwand für die Validierung als eine eigengenutzte Entwicklung.
- An die Bedienbarkeit (Nutzerschnittstelle) werden bei einer "professionellen" Anwendung erheblich höhere Ansprüche gestellt.

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorgehensmodelle

Definition "Prozess" nach IEEE

Eine Folge von Schritten die zu einem definierten Zweck ausgeführt werden

- Beispielsweise der Softwareentwicklungsprozess
- Um Operationen auf Daten auszuführen

Definition "Softwareentwicklungsprozess" nach IEEE

Der Prozess bei dem die Bedürfnisse von Nutzern in ein Softwareprodukt übersetzt werden. Der Prozess beeinhaltet

- das Übersetzen der Bedürfnisse in konkrete Anforderungen,
- das Überführen der Anforderungen in einen Entwurf,
- die Implementierung des Entwurfs in Quelltext,
- das Testen des Quelltextes,
- die Installation und den Betrieb der implementierten Software.

- Softwareprozesse variieren je nach Organisation
- kein Prozess ist perfekt
- Folge: Ergebnisse unterscheiden sich situationsbedingt

Geben Sie

- 1** Beispiele für unterschiedliche Softwareprozesse
- 2** Gründe für diese Unterschiede

Warum ist es schwierig Softwareentwicklungsprozesse zu automatisieren?

Übung 2.1 - Lösung

Beispiele für unterschiedliche Softwareprozesse

1 Planungsgetriebene Prozesse

- 1 Sequenziell
- 2 Nebenläufig
- 3 Inkrementell

2 Agile Prozesse

- 3 ...

Gründe für diese Unterschiede

- 1 Detailgrad der Anforderungen
- 2 Teamstruktur
- 3 Planbarkeit des Softwareprodukts
- 4 Time-to-Market
- 5 Art der Software die entwickelt wird
- 6 Kundentyp
- 7 ...

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Warum ist es schwierig Softwareentwicklungsprozesse zu automatisieren?

- 1 Anforderungen oft nicht final
- 2 Komplexe Systeme sehr schwer zu testen
- 3 Große Systeme besitzen viele Schnittstellen
- 4 ...

Modell

- Abstrakte Abfolge von Schritten
- Dient beliebig vielen Prozessen als Grundlage für konkretes Vorgehen
- Ist ein Muster für eine bestimmte Vorgehensweise

Prozess = Gegenstand des Modells

- Tatsächlich ausgeführte Abfolge von Schritten
- Jeder Schritt produziert ein konkretes Ergebnis
- Ist das Projekt

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Modell

- Theaterstück
- Musik-CD
- Applikation
- Klasse
- Vorgehensmodell
- Prozessmodell

Gegenstand

- Aufführung
- Einmalige Wiedergabe
- Ausführung der Applikation
- Objekt
- Projektablauf
- Projekt (inkl. Organisation)

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Abbildungsmerkmal

- Ein Modell ist immer ein Abbild des Originals
- dass
 - Struktur (z.B. Aufbau eines Hauses),
 - Verhalten (z.B. Schiffsmodell im Strömungskanal)
 - oder Funktionsweise (z.B. Modellauto dass fährt) des Originals abbildet.

Verkürzungsmerkmal

- Es enthält die relevanten Eigenschaften wie
 - detaillierter Skelettaufbau des Menschen für Ärzte
 - oder die Beschreibung der Proportionen des Menschen für Schneider

Pragmatisches Merkmal

- Es ist zugeschnitten auf den Untersuchungszweck und kann damit unter bestimmten Bedingungen das Original ersetzen

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software wird auf unterschiedliche Arten repräsentiert
(Software Modelle)

- Spezifikation
- Entwurf
- Diagramme
- Code
- Kennzahlen
- Dokumentation

Abläufe bei der Entwicklung von Software werden durch Vorgehens-/Prozessmodelle beschrieben

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Basismodelle

Definition Vorgehensmodell

Darstellung, die weitgehend den Softwareentwicklungsprozess beschreibt und prinzipiell auch Analysen des Prozesses gestattet. Ein Vorgehensmodell muss die Prozessschritte und die dabei verwendeten und entwickelten Resultate explizit beschreiben.

Codierung und Bugfixing finden im Wechsel mit Tests statt

- ohne Analyse
- ohne Spezifikation
- ohne Entwurf

Vorteile

- Schnelle Resultate
- Einfacher Ablauf
- Kein Aufwand für Dokumentation und Kommunikation

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Nachteile

- Schlechte Planbarkeit des Projekts
- Arbeit schwer auf mehrere Personen zu verteilen
- Es fehlen die Anforderungen da nicht erhoben
- Programmstruktur leidet durch häufige Nachbesserung
- Oft fehlende Dokumentation
- Hoher Wartungs- und Pflegeaufwand
- Wissen liegt in den Köpfen der Entwickler

Um die Nachteile zu umgehen werden zusätzliche Aktivitäten benötigt



Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

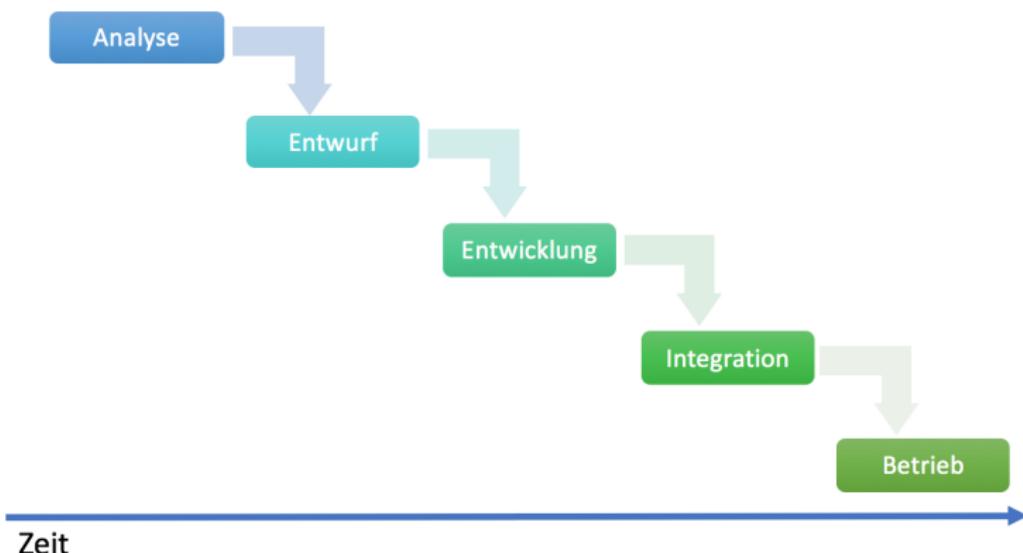
Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Sequenzielles Modell

- Softwareentwicklung wird in Aktivitäten (Phasen) gegliedert
- Aktivitäten werden sequenziell durchlaufen
- Nachfolgeaktivität kann erst dann starten, wenn der Vorgänger vollständig abgeschlossen ist



Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorteile

- leicht verständlich
- geringer Managementaufwand

Nachteile

- Gesamtdauer = Summe aller Aktivitäten
- Fehlende Rückkopplung zwischen den Aktivitäten
- Wird ein Problem in Folgeaktivität erkannt muss von vorn begonnen werden
- Lauffähiges Produkt erst am Ende des Projekts

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

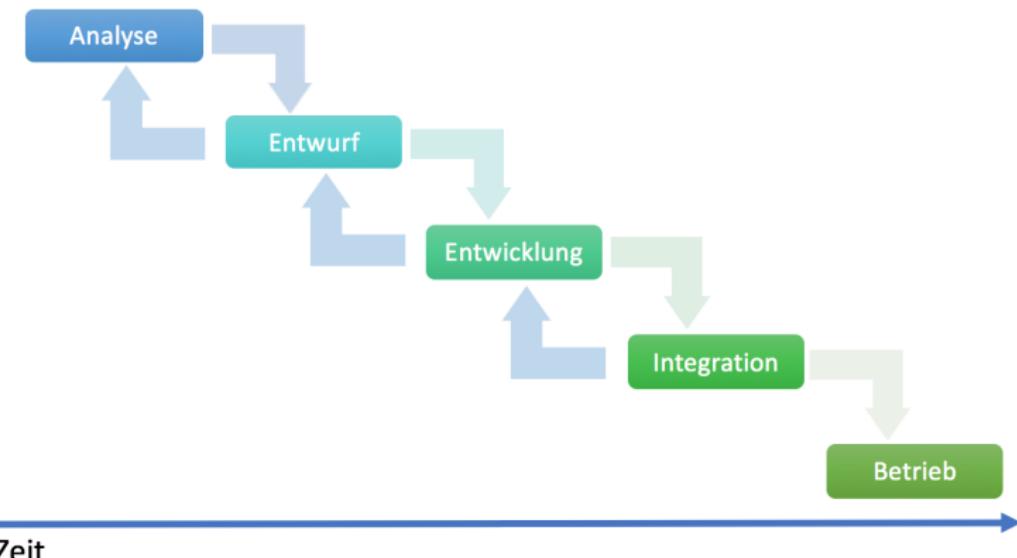
Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Wasserfallmodell als bekanntestes sequentielles Modell

- Sequenzielles Modell mit Rückkopplung
- Jede Aktivität wird vollständig ausgeführt
- Am Ende jeder Aktivität steht ein Dokumentation (Dokumentengetriebenes Modell)
- Benutzer nur in der Analyse beteiligt



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorteile

- leicht verständlich
- geringer Managementaufwand
- Aktivitäten gut dokumentiert

Nachteile

- Es ist nicht immer sinnvoll alle Aktivitäten vollständig auszuführen
- Team ist an die Reihenfolge gebunden
- Die Dokumentation kann wichtiger werden als das eigentliche System
- Es kann nicht flexibel auf Risikofaktoren reagiert werden
- Lauffähiges Produkt erst am Ende des Projekts

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

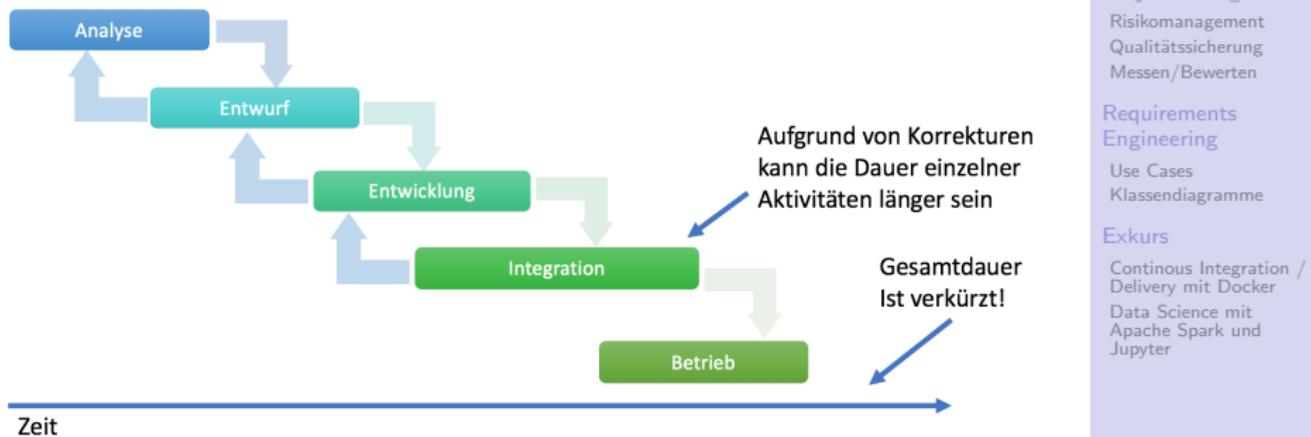
Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Nebenläufiges Modell

- Durch Überlappungen und Rückkopplungen soll die Gesamtzeit reduziert werden
- Nachfolger beginnen sobald Vorgänger die ersten Informationen bereitgestellt haben
- Die Teams arbeiten parallel
- Nachfolger müssen auf neue Informationen der Vorgänger reagieren



Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorteile

- Gute Ausnutzung der Zeit
- Frühzeitige Rückmeldung möglich

Nachteile

- Wichtige Entscheidungen können zu spät getroffen werden
- Hoher Planungs- und Personalaufwand
- Gefahr dass Nachfolger mit unzureichenden Informationen beginnen
- Es kann nicht flexibel auf Risikofaktoren reagiert werden
- Kommunikation zwischen den Teams muss aufrecht erhalten werden

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Sequenzielle Modelle

- Bisher wurde in einem Rutsch ein vollständiges Produkt entwickelt
- Es wurden vor der Implementierung alle Anforderungen im Detail erarbeitet
- Der Kunde ist nur zu Beginn involviert
- Es kann mitunter lange dauern bis der Kunde das Produkt nutzen kann

Problem

- Zu Beginn sind oftmals nicht alle Anforderungen vorhanden
- Der Kunde sollte bereits früh Feedback geben ob das Produkt in seinem Sinne entwickelt wurde

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Anforderungen werden vollständig erfasst und modelliert
- Produkt wird in Ausbaustufen zerlegt
- Jede Ausbaustufe realisiert einen Teil der Funktionalität
- Kunde bekommt sehr früh eine erste Version
- Erfahrungen fließen in die Folgeversionen mit ein
- Jede Version kann in eigenem Projekt entwickelt werden



Teilprodukt (TP) 1 = Version 0

TP 1 + TP 2 = Version 1

TP 1 + TP 2 + TP 3 = Version 2

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitäts sicherung

Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorteile

- Kunde erhält früh und in kurzen Abständen produktionsreife Software
- Nicht-Funktionale Anforderungen werden frühzeitig berücksichtigt
- Durch vollständige Anforderungen kann die Applikation von Beginn an gut strukturiert werden

Nachteile

- Vollständige Anforderungen zu Beginn führen zu einer relativ späten Auslieferung von Version 0
- Modell kann nur verwendet werden, wenn Anforderungen vollständig erfasst sind

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale Prozessmodelle
- Agile Prozessmodelle

Projektmanagement

- Risikomanagement
- Qualitätssicherung
- Messen/Bewerten

Requirements
Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter

- Einsatz wenn zu Beginn nicht alle Anforderungen erfasst werden können
- Es wird mit Kernanforderungen des Kunden angefangen
- Auf dieser Basis wird der Produktkern entwickelt
- Kunde kann früh die erste Version einsetzen
- Aus den Erfahrungen leitet der Kunde weitere Anforderungen ab
- Neue Anforderungen werden in der nächsten Version implementiert
- Zyklus aus praktischer Erprobung und Erweiterung + Verbesserung
- Software wird in Evolutionsstufen entwickelt
- Grundlage der agilen Prozessmodelle

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

Vorteile

- Anforderungen müssen zu Beginn nicht vollständig vorliegen
- Kunde kann sehr früh die erste Version einsetzen und bewerten
- Erfahrungen aus Produktiveinsatz können in nächste Version einfließen
- Durch kurze Entwicklungszyklen kann kurzzeitig auf Änderungen reagiert werden

Nachteile

- Gefahr dass bei Folgeversionen die gesamte Architektur überarbeitet werden muss
- Es besteht die Gefahr dass "evolutionär" nur ein Vorwand für mangelhafte Spezifikation ist

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Wann würden Sie evolutionäre Modelle anwenden und wann sequenzielle?

- Bei Projekten mit offenen Fragen (bzgl. Technologie, Domäne, ...)
- Bei Projekten mit unklaren oder sich ändernden Anforderungen
- Bei sehr komplexen Projekten

Entwickeln Sie eine Applikation. Für Analyse, Entwurf und Entwicklung benötigen Sie jeweils 2 Monate. Sie führen eine nebenläufige Entwicklung durch. Dabei wird jeweils 75% der Vorgängerphase überlappt. Aufgrund des Kommunikations- und Änderungsaufwands verlängern sich die Phasen Entwurf und Entwicklung um jeweils 20%. Wie viel Zeit sparen Sie durch die Nebenläufige Entwicklung ein?

Sequenziell = 2M A + 2M E + 2M C = 6 Monate

Nebenläufig = 2M A + 2.4M E (Start 0.5) + 2.4M C
(Start 1) = ca. 3.5 Monate

Sie arbeiten nun nach dem inkrementellen Vorgehensmodell. Die Analyse benötigt 2 Monate. Sie entwickeln 2 inkrementale (V1 und V2). Für jedes Inkrement benötigen Entwurf und Entwicklung jeweils 1 Monat. Wie viel Zeit wird benötigt um V1, V2 sowie die finale Applikation fertig zu stellen? Wo liegt der Vorteil im Vergleich zur sequenziellen Entwicklung?

$$V1 = 2M A + 1M E + 1M C = 4 \text{ Monate}$$

$$V2 = 1M E + 1M C = 2 \text{ Monate}$$

$$V3 = V1 + V2 = 6 \text{ Monate}$$

Vorteile

- Kunde kann Applikation bereits nach 4 Monaten nutzen
- Inkremeente können jeweils in eigenen Projekten realisiert werden
- ...

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

Entwickeln Sie eine Applikation nun nach dem evolutionären Vorgehensmodell. Für Analyse, Entwurf und Entwicklung benötigen Sie jeweils 1 Monat. Sie entwickeln 2 Versionen (V1 und V2). Wie lange dauert die Entwicklungszeit für V1 und V2 sowie für das ganze Produkt? Wo liegt der Vorteil im Vergleich zur sequenziellen Entwicklung?

$$V1 = 1M A + 1M E + 1M C = 3 \text{ Monate}$$

$$V2 = 1M A + 1M E + 1M C = 3 \text{ Monate}$$

$$V3 = V1 + V2 = 6 \text{ Monate}$$

Vorteile

- Kunde kann Applikation bereits nach 3 Monaten nutzen
- Architektur und Code ist auf die Problemstellung abgestimmt
- ...

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

- Entwicklung einer Anfangsimplementierung
- Benutzer geben zu dieser (konkreten) Implementierung Feedback
- Äußerungen der Benutzer werden in neuer Version des Systems berücksichtigt
- Die Schritte 2 – 3 werden solange durchgeführt bis ein angemessenes System entstanden ist

Prototyp

Provisorisches Softwaresystem (Modell), das erstellt wird, um Anforderungen zu klären oder zu veranschaulichen.

Prototyping

Folge von Prototypen, die bestimmte Systemfunktionen oder -aspekte frühzeitig realisieren oder vortäuschen, so dass der Benutzer:

- den gewünschten Eindruck erhält
- sich mit dem Prototyp auseinandersetzen kann.

Rapid Prototyping

Art von Prototyping bei dem der Fokus auf der Entwicklung von Prototypen zu Beginn des Softwareprozesses an liegt, um so möglichst von Anfang an Feedback vom Nutzer einzuholen.

Prototypen haben die folgende Verwendung:

- Klärung von Anforderungen oder Entwicklungsproblemen
- Durchführung von Experimenten und Sammlung von Erfahrungen

Dabei wird zwischen verschiedenen Arten von Prototypen unterschieden.

Demonstrationsprototyp

- Dient der Auftragsakquisition
- Auftraggeber bekommt ersten Eindruck vom Produkt
- Schnelle Entwicklung in frühem Stadium des Entwicklungsprozesses
- Auf Standards und Best Practices bei der Entwicklung wird verzichtet
- Wegwerfprodukt

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Funktionaler Prototyp

- Unterstützt die Anforderungsanalyse
- Modelliert Ausschnitte der Bedienoberfläche und/oder Teile der Funktionalität
- Kann in Architektur bereits dem Zielsystem entsprechen
- Dient dem Nachweis der technischen Machbarkeit
- Auch für Benutzerfeedback hinsichtlich Software-Ergonomie

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Labormuster

- Modelliert technische Aspekte des Zielsystems
- Experimentiersystem für Entwickler
- Für Machbarkeitsstudien
- In der Regel sind Endnutzer nicht an der Evaluation beteiligt

Pilotsystem

- Realisiert einen abgeschlossenen Bereich des Zielsystems
- Funktionalität und Qualität reichen mindestens für vorübergehenden Produktiveinsatz
- Wird Schrittweise erweitert

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Ein Softwaresystem besteht in der Regel aus verschiedenen Komponenten/Schichten. Aus diesem Grund lassen sich Prototypen auch in horizontale oder vertikale Prototypen unterscheiden.

- Horizontale Prototypen realisieren nur eine Schicht des Systems (beispielsweise Nutzeroberfläche oder Datenbankschicht)
- Vertikale Prototypen implementieren einen funktionalen Teilbereich durch alle Schichten hindurch (End-2-End)

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

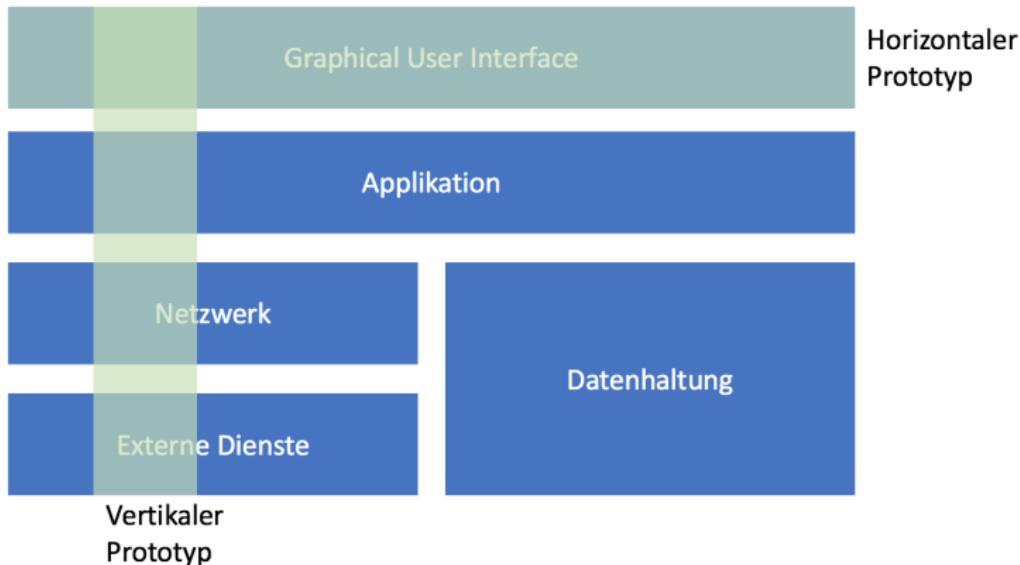
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

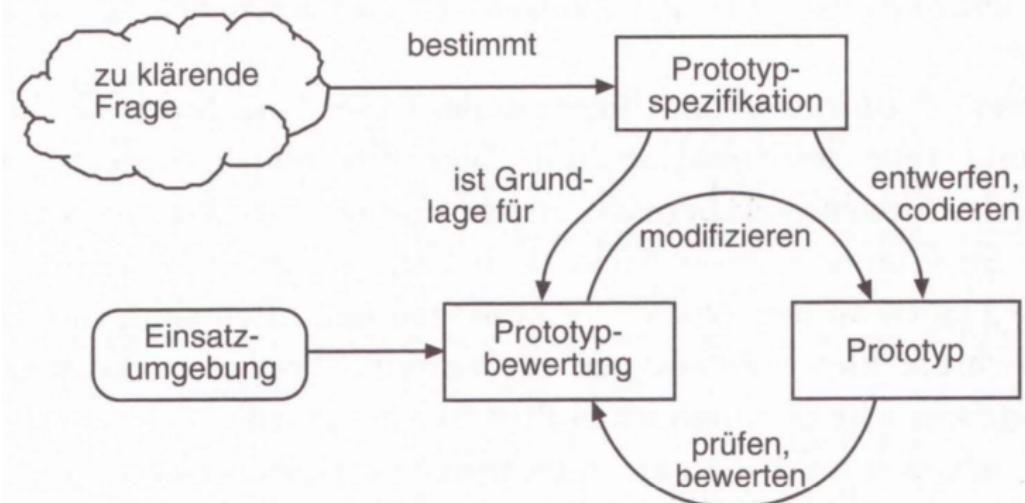
Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



- Für sehr interaktive Systeme
- Für kleine/mittlere oder Teile großer Systeme
- Für Systeme mit kurzer Lebensdauer
- Für Systeme, die schlecht vorab planbar sind

Prototyping wird selten als alleinstehendes Vorgehensmodell verwendet. Vielmehr findet es im Rahmen anderer Vorgehensmodelle statt.

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Welche Vor- und Nachteile hat die Verwendung von
Prototypen?

Vorteile

- Reduziert das Entwicklungsrisiko
- Missverständnisse in den Anforderungen fallen frühzeitig auf
- Fehlende Anforderungen werden identifiziert wodurch die Planung optimiert wird
- Fördert die Kreativität

Nachteile

- Die entstehenden Applikationen sind oftmals schlechter strukturiert
- Wegwerf-Prototypen werden of nachträglich als evolutionär entwickeltes System deklariert und produktiv eingesetzt

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Generisches Vorgehensmodell
- Bietet Anleitung um unter Berücksichtigung von spezifischen Risikofaktoren ein auf das Projekt zugeschnittenes Vorgehensmodell zu entwickeln
- Mit Blick auf die Risiken wird für jede Phase ein eigenes Vorgehensmodell verwendet

Für jede Phase werden 4 Schritte durchlaufen.

Schritt 1:

- 1** Identifikation der Ziele des Teilprodukts (Leistung, Funktionalität...)
- 2** Alternative Möglichkeiten Aufzeigen um das Teilprodukt zu realisieren (Entwurf 1, Entwurf 2, Kauf, Wiederverwendung)
- 3** Randbedingungen beachten (Schnittstellen, Zeit, Budget ...)

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Schritt 2:

- 1 Alternativen evaluieren
- 2 Risiken identifizieren
- 3 Zeichnen sich Risiken ab, werden geeignete Maßnahmen ergriffen um sie zu vermeiden
- 4 Prototypen entwickeln

Schritt 3:

- 1 Unter Berücksichtigung der verbleibenden Risiken wird geeignetes Vorgehensmodell gewählt
- 2 Entwicklung

Schritt 4:

- 1 Nächsten Zyklus planen
- 2 Review der vorhergehenden Schritte

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /

Delivery mit Docker

Data Science mit

Apache Spark und

Jupyter

Spiralmodell



Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Fläche der Spirale steht für angefallene Kosten
- Winkel der Spirale zeigt Fortschritt der Entwicklung im aktuellen Durchlauf

Vorteile

- Risiken frühzeitig berücksichtigt
- Modell ist sehr flexibel
- Andere Vorgehensmodelle können integriert werden
- Frühe Erkennung und Eliminierung von Fehlern
- Unterstützt Wiederverwendung da Alternativen berücksichtigt werden
- Anpassung der Entwicklung durch Erfahrungen die im Zyklus t-1 gewonnen wurden

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Nachteile

- Ständiger Wechsel des Vorgehensmodells ist wenig Praktikabel
 - hoher Managementaufwand
 - Mitarbeiter müssen sehr flexibel seinem
 - Detailwissen und Erfahrung verschiedener Modelle sind Voraussetzung
 - Nicht für kleine bis mittlere Projekte geeignet
 - Oftmals fehlende Erfahrung im Risikomanagement
 - Geplant wird über einen Zyklus, End-2-End Planung nicht möglich

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Nennen Sie 5 mögliche Gründe warum Projekte
(insbesondere Softwareprojekte) scheitern.

- Projektziel/Strategie nicht klar kommuniziert oder gar definiert
- Verwendung unausgereifte Technologien
- Kommunikationsdefizite innerhalb des Teams
- Kommunikationsdefizite gegenüber dem Kunden
- Kein klares Projektvorgehen definiert
- Fehlende Transparenz bezüglich des Projektfortschritts
- Schlechte Verteilung von Skills und Erfahrung
- Unternehmenspolitische Hürden
- ...

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

In einem Softwareprojekt können immer wieder Führungsprobleme auftreten. Was können mögliche Ursachen dafür sein? Betrachten Sie die Frage aus dem Blickwinkel des Projektleiters sowie aus Perspektive des Mitarbeiters.

Übung 2.8 - Lösung

Sicht des Projektleiters

- Fortschritte werden nicht korrekt Kommuniziert / Fehlende Transparenz
- Komplexität einzelner Arbeitspakete für Nicht-Entwickler schwer bewertbar
- Fehlender Fokus der Entwickler
- Projektressourcen werden abgezogen
- ...

Sicht des Mitarbeiter

- PM bringt zu wenig technisches Wissen mit
- Auswirkungen von ad-hoc Änderungen werden vom Management oft unterschätzt / PM hat kein Durchsetzungsvermögen
- ...

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

Monumentale Prozessmodelle

Definition Prozessmodell

Während Vorgehensmodelle den Kern bilden, ergänzen Prozessmodelle die Vorgehensmodelle um Organisationsstrukturen für Projektmanagement, Qualitätssicherung, Dokumentation sowie Konfigurationsverwaltung.

Monumental

- Prozessmodelle mit sehr umfangreicher Beschreibung
- Dokumentation von zentraler Bedeutung und umfangreich
- Planung ist fest vorgegeben

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

- Aktivitäten werden von Mitarbeitern durchgeführt
- Die Kenntnisse/Fähigkeiten die als Voraussetzung dienen werden durch Rollen beschrieben
- Die Durchführung wird genauer spezifiziert
 - Weitere durchzuführende Aktivität?
 - Rollenzuordnung
 - Zu verwendende Artifakte
 - Zu erstellende Artifakte
 - Zu beachtende Konventionen, Methoden, Richtlinien
 - Einzusetzende Werkzeuge

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

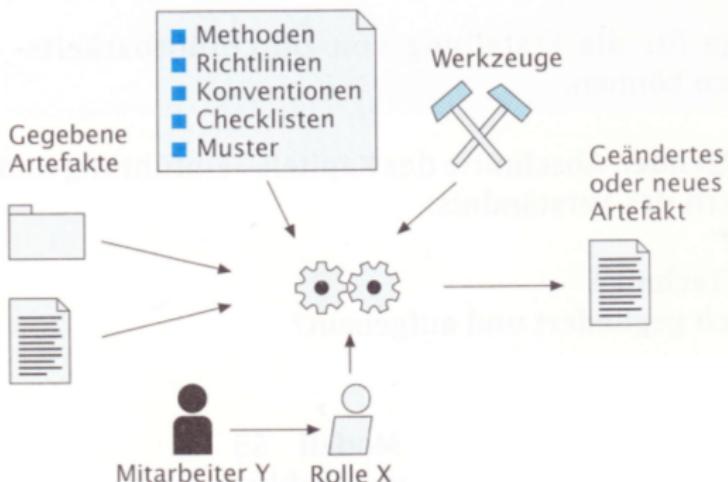
Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter



Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale Prozessmodelle**
- Agile Prozessmodelle

Projektmanagement

- Risikomanagement
- Qualitäts sicherung
- Messen/Bewerten

Requirements Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continuous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter

- Erweiterung des Wasserfallmodells
- Integration einer Qualitätssicherung
- Erzeugte Teilprodukte werden Verifikation und Validierung unterzogen
 - Verifikation: Prüfung ob das SW-Produkt mit der Spezifikation übereinstimmt ("Am I building the product right?")
 - Validierung: Eignung des Produkts für den vorgeschriebenen Einsatzzweck ("Am I building the right product?")

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

**Monumentale
Prozessmodelle**

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitäts sicherung

Messen/Bewerten

Requirements
Engineering

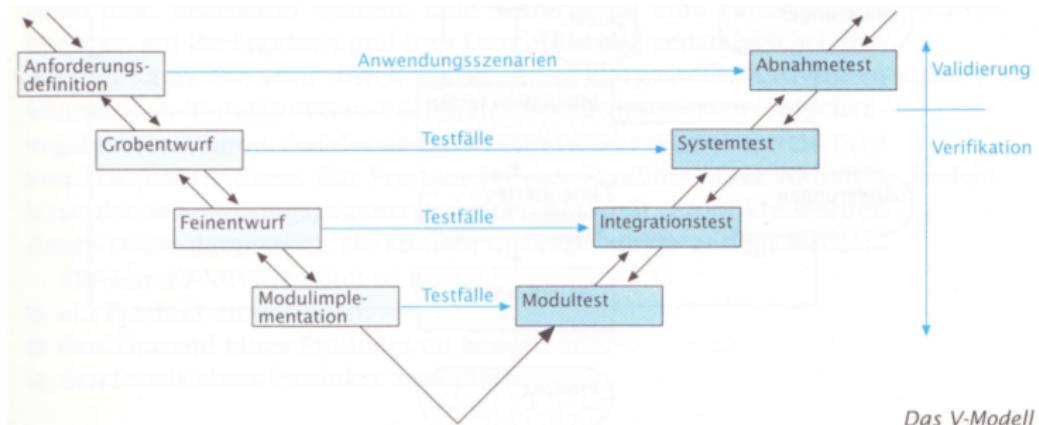
Use Cases

Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter



Das V-Modell

- Regelt “wer”, “wann”, “was” in einem Projekt zu tun hat
- Unterscheidet 4 verschiedene Projekttypen zur Abbildung von Auftraggeber und Auftragnehmer
- Charakteristische Projekteigenschaften werden berücksichtigt

Projektrolle	Auftraggeber	Auftragnehmer	Auftraggeber/Auftragnehmer	Auftraggeber/Auftragnehmer
Projekttyp	Systementwicklungsprojekt (AG)	Systementwicklungsprojekt (AN)	Systementwicklungsprojekt (AG/AN)	Einführung und Pflege eines organisationsspezifischen Vorgehensmodells
Projektgegenstand [Projektmerkmal]	HW-System	SW-System	HW- und SW-System / Eingebettetes System	Systemintegration

- Für jeden Typen bietet V-Modell angepasste Varianten an
- Variante bestimmt dann die Durchführungsstrategie



Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

- Projektdurchführungsstrategie legt Reihenfolge der zu erstellenden Produkte und durchzuführenden Aktivitäten fest
- Grundlage für den Projektplan
- Es stehen 11 Strategien zur Auswahl

Für Entwicklungsprozess aus AN-Sicht mit hohem Realisierungsrisiko werden z.B. folgende Strategien vorgeschlagen:

- Inkrementelle Entwicklung
- Komponentenbasierte Entwicklung
- Agile Entwicklung
- ...

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

- Produkte sind Ergebnisse und Zwischenergebnisse eines Projekts
- 4 Produkte werden für alle Projekte verpflichtend vorgeschlagen:
 - Projekthandbuch
 - Projektplan
 - QS-Handbuch
 - Produktbibliothek
- Jedes Produkt wird durch eine Aktivität erstellt

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

- Zusammengehörige Produkte und Aktivitäten werden als Disziplinen bezeichnet
- Es werden 13 Disziplinen unterschieden und in die drei Bereiche Projekt, Entwicklung und Organisation unterteilt

Projekt	Entwicklung	Organisation
Planung und Steuerung	Anforderungen und Analyse	Prozessverbesserung
Berichtswesen	Systemelemente	
Konfigurations- und Änderungsmanagement	Systementwurf	
Prüfung	Logische Konzeption	
Ausschreibung / Vertragswesen	Logistikelemente	
Angebot / Vertragswesen	Systemspezifikation	

Tabelle: Aktivitäten der Disziplin “Planung und Steuerung”

Planung und Steuerung

-
- Projektfortschrittsentscheidung herbeiführen
 - Projekthandbuch erstellen
 - QS Handbuch erstellen
 - PM-Infrastruktur einrichten
 - Schätzung durchführen
 - Risiken managen
 - Projekt planen
 - Arbeitsauftrag vergeben
 - Kaufm. Projektkalkulation durchführen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

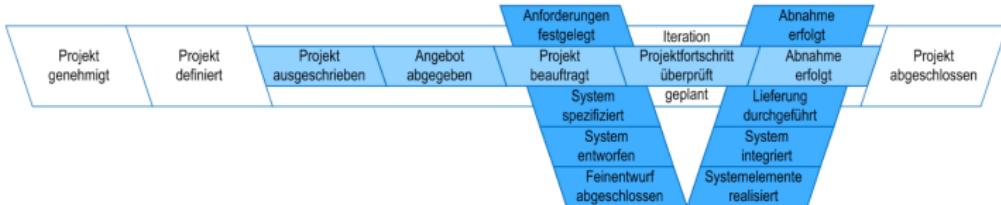
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

V-Modell Xtreme Tailoring



- Managementaspekte (weiße Entscheidungspunkte) sind in jeder Projektdurchführungsstrategie enthalten
- Auftraggeber-/Auftragnehmer-Schnittstelle (hellblaue Entscheidungspunkte) betreffen die Schnittstelle zwischen Auftraggeber und Auftragnehmerprojekten
- Systemerstellung (dunkelblaue Entscheidungspunkte) beziehen sich auf die Systemerstellung

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- V-Modell XT wird aus Vorgehensbausteinen zusammen gesetzt
- Vorgehensbaustein fasst die für konkrete Aufgabe erforderlichen Disziplinen zusammen (z.B. alle Disziplinen die für Änderungsmanagement)
- Auch die an Produkt mitwirkenden Rollen werden durch Vorgehensbaustein festgelegt
- Den V-Modell Kern bilden folgende Bausteine:
 - Projektmanagement
 - Qualitätssicherung
 - Problem- und Änderungsmanagement
 - Konfigurationsmanagement
- Abhängig von Projektmerkmalen kommen weitere Bausteine dazu

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit
Apache Spark und
Jupyter

Vorteile

- Öffentlich zugänglich
- Werkzeuge und Dokumentation existieren
- V-Modell XT kann als Baukasten verwendet und an Projektsituation angepasst werden
- Enthält viele Produktvorlagen

Nachteile

- Hohe Komplexität erfordert hohen Schulungsaufwand
- Generiert hohe Menge an Artefakten
- Managementaufwand ist hoch

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitäts sicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Agile Prozessmodelle

Agil

- Prozessmodell mit kleinem bis mittlerem Umfang beschrieben
- Geringer Aufwand für Dokumentation
- Flexible Prozessabläufe
- Teammitglieder, Kunde, Code liegen im Fokus

Agiles Manifest

- Einzelpersonen und Interaktion >Prozesse und Werkzeuge
- Laufende Systeme >Umfassende Dokumentation
- Zusammenarbeit mit Kunden >Vertragsverhandlungen
- Reaktion auf Änderung >Verfolgen eines Plans

- Formale Aspekte der SW werden auf Minimum reduziert
 - Verzicht auf Dokumentation
 - Fokus auf lauffähigen Code
- Statt auf detailliertem Prozessmodell basiert XP auf
 - Werten
 - Prinzipien
 - Praktiken
- Testfälle (Modul- und Akzeptanztests) ersetzen Spezifikation und Entwurf

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Werte

- Einfachheit: Einfache Lösungen die leicht zu verstehen und schnell zu realisieren sind.
- Kommunikation: Persönlich und direkt, Dokumente für Informationsaustausch zweitrangig
- Feedback: Schnelle Rückmeldung über Ergebnisse an das Team
- Mut: Zur Lücke und zur Kommunikation

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Prinzipien

- Unmittelbares Feedback
- Einfachheit anstreben
- Inkrementelle Veränderung
- Veränderung wollen
- Qualitätsarbeit
- Lernen lehren
- Geringe Anfangsinvestition
- Auf Sieg spielen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Gezielte Experimente
- Offene, aufrichtige Kommunikation
- Instinkte der Teammitglieder nutzen und nicht dagegen arbeiten
- Verantwortung übernehmen
- An Projektgegebenheiten anpassen
- Mit leichtem Gepäck reisen
- Ehrliches Messen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Praktiken

- Kunde ist Teil des Teams und vor Ort
- User Stories
- Kurze Releasezyklen
- Akzeptanztests
- Pair Programming
- Testgetriebene Entwicklung
- Gemeinsame Verantwortung

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Continuous Integration
- Nachhaltiges Tempo
- Offene Räume
- Planungsspiel
- Einfacher Entwurf
- Refactoring
- Metapher

Vorteile

- Testgetrieben
- Kundennah
- Dokumentation besteht aus Code + User Stories und ist damit offen für Änderungen
- Paarweises Programmieren

Nachteile

- Wiederverwendung oft nicht gegeben da sehr projektspezifischer Code
- Hochqualifizierte Teams gefordert
- Sehr hoher Anspruch an Kommunikation

- Managementkonzept dass nicht nur auf SW-Entwicklung beschränkt ist
- Setzt auf Selbstorganisation der Entwickler
- Ansatz ist empirisch, inkrementell und iterativ
- Basiert auf Transparenz, Überprüfung, Anpassung
- Vorgehensmodell wird durch wenige einfache Regeln definiert (easy to learn, hard to master)

Als agiler Prozess kommt Scrum mit einigen wenigen Dokumenten/Artefakten aus (nach Scrum Guide)

- Product Backlog, enthält alle Anforderungen in Form von User Stories nach Priorität absteigend geordnet
- Sprint Backlog, enthält alle Anforderungen in Form von User Stories die im Sprint umgesetzt werden
- Inkrement, fertiges Produkt als Sprintergebniss
- Definition of Done, gemeinsames Verständnis darüber was es heißt eine Story abgeschlossen zu haben

Daneben existieren außerdem Sprint Ziele, Burndown Charts, Impediment Backlog, Epic Board ...

Scrum definiert drei Rollen

- Product Owner
 - Kennt fachliche und technische Anforderungen
 - Pflegt, erweitert und priorisiert das Product Backlog
 - Beurteilt die Sprintergebnisse
 - Entwickelt das Produkt weiter, hat eine klare Vision
- Scrum Master
 - Moderator, Coach, Serving Leader
 - Sorgt für die Einhaltung der Scrum-Regeln
 - Hilft bei methodischen Problemen
 - Beseitigt Hindernisse
 - Treibt agile Vorgehensweise über Projektgrenzen hinaus voran
- Projekt Team
 - 4-12 Teammitglieder
 - Organisiert sich selbst
 - Keine Hierarchien
 - Kennt nur Entwickler
 - Unterschiedliche Kompetenzen

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale

Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit

Apache Spark und
Jupyter

Scrum definiert eine Reihe von Events

- Daily Scrum, 15 Minuten, 3 Fragen: Was habe ich gestern gemacht? Was mache ich heute? Was blockt mich?
- Sprint Review, Vorstellung der Sprintergebnisse
- Sprint Retrospective, Reflektion des Sprints, was lief gut, was schlecht
- Sprint Planning, erstellen des Sprint-Backlogs, was wird wie erreicht

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale

Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

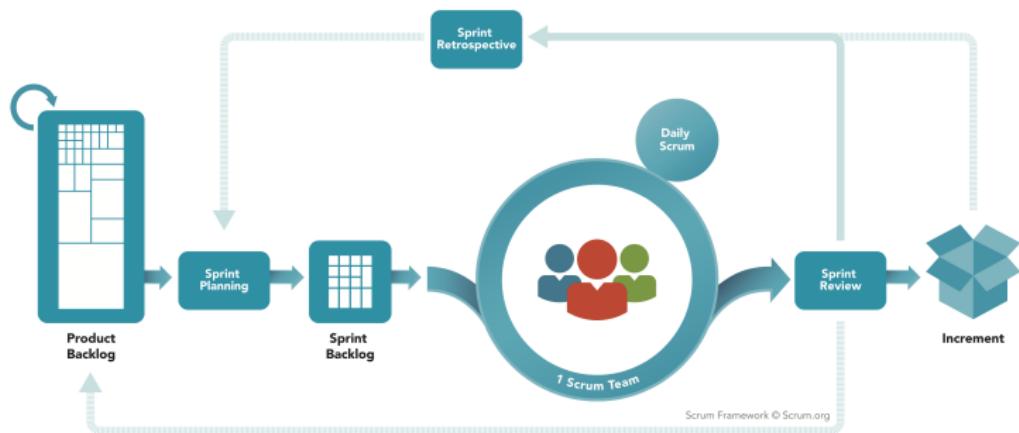
Exkurs

Continous Integration /
Delivery mit Docker

Data Science mit

Apache Spark und

Jupyter



Vorteile

- Leichtgewichtig
- Klar definierte Regeln und Abläufe
- Fördert Kreativität durch selbstorganisierte Teams
- Leicht zu lernen

Nachteile

- Wiederverwendung oft nicht gegeben da sehr projektspezifischer Code
- Schwierig wenn Teams weltweit verteilt
- Regelwerk ist unflexibel
- Schwer zu meistern

Kanban definiert sich durch vier Elemente

- Arbeit wird genommen, nicht gegeben
- Mengen werden limitiert
- Informationen werden veröffentlicht
- Abläufe werden kontinuierlich verbessert

- Kanban-Board ist zentrales Element
- Besteht aus frei definierbaren Spalten
- Visualisiert den Entwicklungsprozess
- User-Stories/Arbeitspakete werden eigenständig vom Team dem Prozess nach abgearbeitet
- Wann eine einzelnen Phase abgeschlossen ist wird in der Definition of Done festgelegt
- Jede Phase hat ein Limit das die maximale Anzahl aufgenommener Aufgaben angibt
- Limit macht Engpässe transparent

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

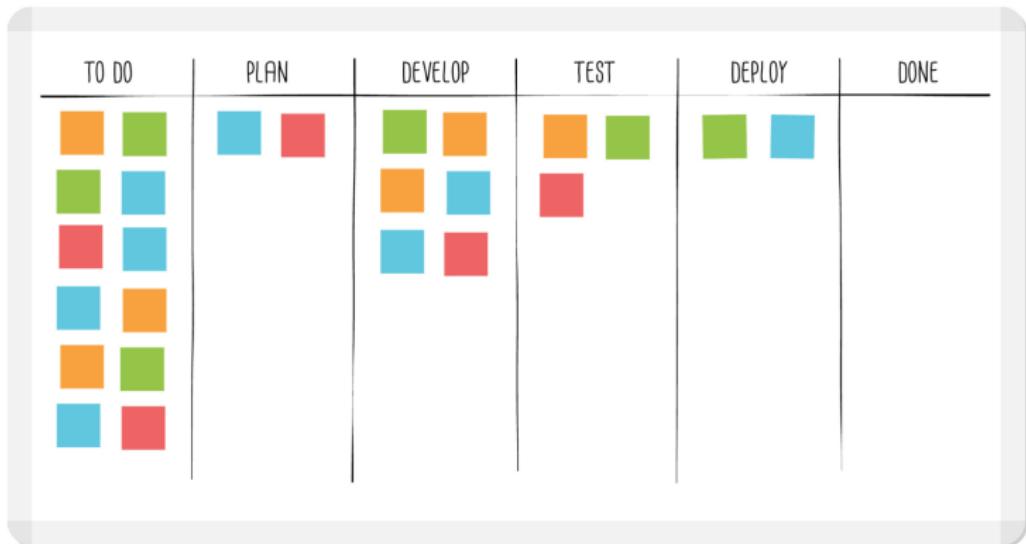
Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Kanban

Mark Keinhrörster



User Story

Defect

Task

Feature

Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale
- Prozessmodelle

Agile Prozessmodelle

Projektmanagement

- Risikomanagement
- Qualitätssicherung
- Messen/Bewerten

Requirements
Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continuous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter

Metriken dienen als Grundlage der kontinuierlichen Verbesserung

- Cumulative Flow Diagram = Summe aller Aufgaben pro Phase in Abhängigkeit der Zeit
- Anzahl der Tickets die Work-in-Progress sind
- Durchsatz an Tickets pro Woche
- Durchschnittliche Zeit die ein Ticket für gesamten Durchlauf benötigt

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorteile

- Leichtgewichtig
- Sehr leicht zu lernen
- Forciert einen ausgeglichenen Arbeitsfluss
- Probleme werden direkt visualisiert

Nachteile

- Für große Projekte unübersichtlich
- Nicht auf Einzelprojekte/Neuprojekte anwendbar

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Projektmanagement

Es gibt einige, allgemeine Erfolgskriterien die alle Projekte teilen

- Software wird innerhalb der definierten Zeit fertiggestellt
- Kosten liegen innerhalb der Budgetplanung
- Gelieferte Software entspricht der Erwartung des Kunden
- Entwicklungsteam arbeitet kohärent und effizient zusammen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Softwareprojekte haben bestimmte Charakteristika die sie von Projekten in anderen Bereichen unterscheiden

- Software ist intangibel
- Große SW-Projekte sind in Technologie, Vorgehen, Organisation einzigartig
- Prozesse und Vorgehensmodelle sind flexibel und organisationsspezifisch

- Größe der Organisation
- Kundentyp
- Komplexität der Software
- Softwaretyp
- Unternehmenskultur
- Softwareentwicklungsprozesse

- 1 Projektplanung**
- 2 Risikomanagement**
- 3 Personalführung**
- 4 Berichterstattung**
- 5 Projektierung**

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Risikomanagement

Risiko

Ein Problem, das noch nicht eingetreten ist, aber wichtige Projektziele oder Projektergebnisse gefährdet, falls es eintritt. Ob es eintreten wird kann nicht sicher vorausgesagt werden.

- In jedem Projekt treten Probleme auf die die Projektziele gefährden
- Risikomanagement versucht mögliche Probleme frühzeitig zu identifizieren und Maßnahmen einzuleiten
- Risikomanagement ist eine kontinuierliche Aktivität und umfasst
 - Identifikation von Risiken
 - Analyse und Bewertung der Risiken
 - Planung von Gegenmaßnahmen
 - Risikoüberwachung

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätsicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Eine Risikoeinordnung kann sein

- Projektrisiken die den Projektplan oder -ressourcen beeinflussen
- Produktrisiken die Qualität oder Effizienz der Applikation mindern
- Geschäftsrisiken den zugrundeliegenden Business Plan der Applikation beeinflussen

- Es kann auch zwischen Kernrisiken und projektspezifischen Risiken unterschieden werden.
- Projektspezifische Risiken können beispielsweise im Rahmen eines moderierten Workshops erhoben werden

Überlegen Sie, welche Kernrisiken für einen Großteil der Soll-Ist-Abweichungen in Softwareprojekten verantwortlich ist.

- Unklare bzw. sich laufend ändernde Projektziele und Anforderungen die grundlegende Änderungen am Quellcode erfordern
- Unkorrekter Zeitplan, z.B. die Dauer der Testphase wurde unterschätzt oder gar unterschlagen
- Mitarbeiterfluktuation im Projektteam
- Mangelnde Skills der Mitarbeiter
- Fehlende Unterstützung im Management
- Organisatorische Änderungen wie beispielsweise wechselnde Projektverantwortliche
- Grundlegend falsch gewählte Technologien
- ...

Risikowert

$$\text{Risikowert} = p * K$$

- p = Eintrittswahrscheinlichkeit
 - K = Kosten die im Schadensfall entstehen
-
- Risiken mit $p > 0.5$ sollten als sicheres Ereignis angesehen werden
 - Jeder Risikowert kann als Punkt in einem Risikodiagramm eingesetzt werden
 - Ein Risikodiagramm besitzt auf beiden Achsen eine logarithmische Skalierung
 - Kleine Risikowerte müssen nicht kontrolliert werden

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

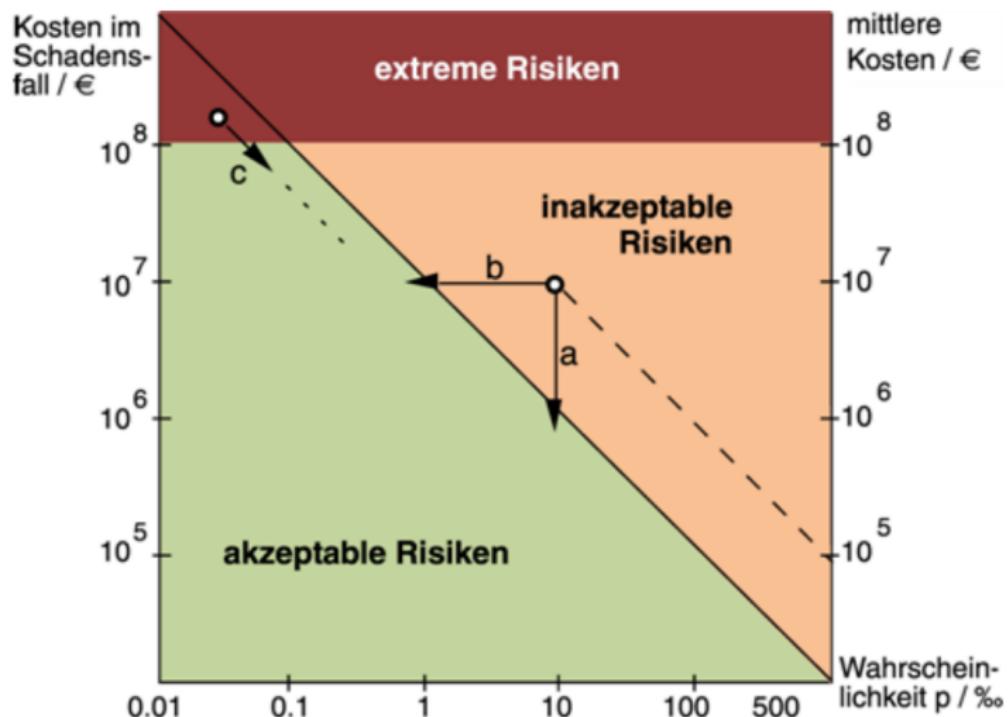
Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Risikodiagramm

Das dargestellte Risikodiagramm bewertet Risikowerte bis 10000 Euro als akzeptabel



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Für Punkte unter der Diagonalen sind keine Aktionen erforderlich
- Punkte über der Diagonalen müssen nach links oder unten verschoben werden
- Senkung der Kosten bei Risikoeintritt verschiebt den Punkt nach unten (Pfeil a)
- Senkung der Wahrscheinlichkeit verschiebt den Punkt nach links (Pfeil b)
- Alle Maßnahmen führen zu zusätzlichen sicheren Kosten die deutlich geringer sind als der mögliche Schaden
- Oben liegen extreme Risiken deren Schaden nicht beherrschbar ist (z.B. Firmenbankrott)
- Bei einem solchen Extremrisiko kann eine Versicherung in Erwägung gezogen werden die aus großem Eintrittsschaden eine kleinere sichere Zahlung macht (Pfeil c)

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Alternativ kann man mit einer diskreten Bewertung arbeiten

Wert (p)	Eintrittswahrscheinlichkeit
1	Unwahrscheinlich
2	Denkbar
3	Wahrscheinlich

Wert (k)	Schadensausmaß
1	Gering
2	Beträchtlich
3	Groß

Risikomatrix

Die Risiken können entsprechend klassifiziert werden um daraus eine Risikomatrix abzuleiten.

Groß	3	6	9
Beträchtlich	2	4	6
Gering	1	2	3
	Unwahrscheinlich	Denkbar	Wahrscheinlich

Auch hier ist zu bestimmen welche Risikowerte kontrolliert werden müssen.

- Zur Risikobegrenzung sollten Zeit-/Geldreserven eingeplant werden
- Die Rückstellung sollte dem Risikowert entsprechen
- Präventivmaßnahmen sind aktiv
 - Senken der Eintrittswahrscheinlichkeit
 - Schaden auf eine vertretbare Höhe beschränken
- Notfallmaßnahmen im Schadensfall sind reaktiv
- Präventive und reaktive Maßnahmen müssen im Projektplan berücksichtigt werden

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

In ihrem Projekt haben Sie den wegang eines wichtigen Mitarbeiters als Risiko identifiziert. Nennen Sie eine Präventivmaßnahme, um die Eintrittswahrscheinlichkeit dieses Risikos zu senken sowie eine Präventivmaßnahme um die Höhe des Schadens zu senken. Was ist eine mögliche Notfallmaßnahme?

Eintrittswahrscheinlichkeit senken:

- Horizontale/Vertikale Karriereanreiz schaffen

Schaden senken:

- Wissensmanagement/Projektkonzept für schnelle Einarbeitung etablieren

Notfallmaßnahme:

- Projektscope verkleinern

- Risiken ändern sich während des Projektverlaufs
 - Schwächen ab
 - Verstärken sich
 - Sind nicht mehr existent
 - Entstehen
- Risikosituation musst stetig neu bewertet werden
- Regelmäßige Projektsitzungen sind unabdinglich
- Risiken müssen offen kommuniziert werden dürfen

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Qualitätssicherung

Qualität

Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit, die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen.

- Im Software Engineering gibt es zwei Sichtweisen auf die Qualität
 - Prozessqualität (z.B. bewertbar durch CMMI)
 - Produktqualität
- Produktqualität umfasst Wartbarkeit und Brauchbarkeit
- Geforderte Qualitätsanforderungen müssen definiert und die Einhaltung dieser Anforderungen geprüft werden

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Qualitätsmanagement

Aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität, die üblicherweise das festlegen der Qualitätspolitik und -ziele, die -planung, die -lenkung die -sicherung und die Qualitätsverbesserung umfassen.

- Unter Qualitätsmanagement werden die managementbezogenen Tätigkeiten zusammengefasst
- Unter Qualitätssicherung werden die technisch orientierten Aktivitäten zusammengefasst

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Es werden drei Maßnahmen zur Qualitätssicherung unterschieden

- Organisatorische Maßnahmen

- Systematische Entwicklung und QS
- Zeitplanung für Testing und Bugfixing
- Rollen / Verantwortlichkeiten für QS
- Vorgabe von Richtlinien, Standards, Checklisten

- Konstruktive Maßnahmen

- Fehler und schlechte Qualität von Beginn an vermeiden
- "Es kann keine Qualität in ein Produkt hineingeprüft werden"

- Analytische Maßnahmen

- Software-Tests
- Metriken
- Audits
- Fehler und Mängel in den Arbeitsergebnissen erkennen

Organisatorische und konstruktive Maßnahmen sind hilfreicher als analytische Maßnahmen.

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Nennen Sie konstruktive Maßnahmen die dabei helfen Mängeln bei der Software-Entwicklung aus dem Weg zu gehen.

- Passendes und vor allem korrekt eingesetztes Vorgehensmodell definieren
- Eingeplante Code-Reviews
- Pair-Programming
- Definierte Quality Gates mit klaren abnahmekriterien
- Einführung von Continuous Integration und Continuous Deployment
- Verwendung von typisierter Programmiersprache
- Schulungen
- Konfigurationsmanagement
- Passende Werkzeuge/Methoden etablieren
- ...

- Fehler die früh in der Entwicklung entstehen und unentdeckt bleiben können zu Summationseffekten in späteren Phasen führen
- Folgende Ziele sollten verfolgt werden:
 - Kein Fehler machen (z.B. durch konstruktive Maßnahmen)
 - Fehler die dennoch gemacht wurden möglichst früh entdecken und beseitigen

Es wird ein Software-Test vorgenommen. Ist es ratsam, jeden entdeckten Fehler sofort zu korrigieren oder gibt es Gründe die Korrektur von der Prüfung zu trennen?

- Korrektur ist Änderung des Artifakts und bedarf eines erneuten Deployments
- Korrektor kann langwierig sein (von wenigen Minuten bis zu mehreren Tagen)
- Einfache Fehler mit geringen Effekten haben ggf. weniger Priorität
- ...

Es sollte stets zwischen Test und Korrektur getrennt werden.

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Qualität in der agilen Softwareentwicklung bedeutet Codequalität
- Fokus liegt auf Praktiken wie Refactoring und testgetriebener Entwicklung
- QM ist formlos und basiert auf qualitätsgtriebener Projektkultur
- Jedes Teammitglied ist für SW-Qualität im Projekt verantwortlich

■ Check before check-in

Entwickler organisieren selbst Code-Review bevor Code gemerged wird

■ Never break the build

Es wird kein Code gepushed der das System als Ganzes zerstört. Wer einen fehlerhaften Build pushed muss die Probleme mit höchster Priorität beheben

■ Fix problems when you see them

Der Code des Systems gehört dem ganzen Team.

Findet ein Entwickler einen Fehler kann er ihn selbst beheben ohne auf den Autor verweisen zu müssen.

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

■ Testgetriebene Entwicklung

Test first programming. Es wird zuerst der Testfall geschrieben und im Anschluss gegen ihn entwickelt

■ Continuous Integration Kontinuierliche Integration von Codeänderungen mit anschließender Verifikation durch Bau des Artefakts

■ Continuous Delivery Möglichkeit kontinuierlich Codeänderungen auf QA/Produktionssysteme in Betrieb nehmen zu können

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

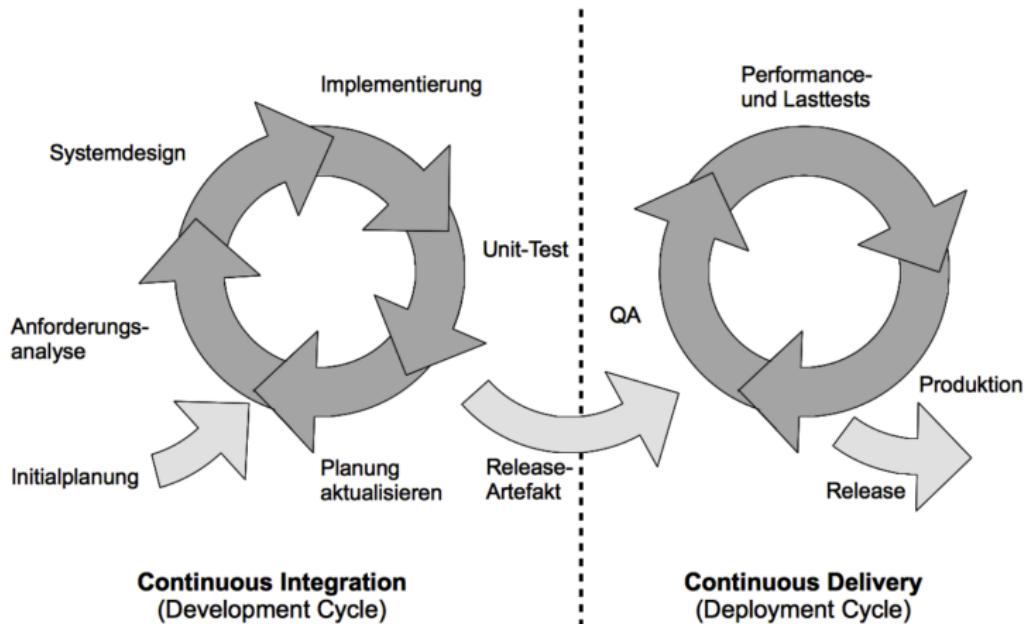
Risikomanagement
Qualitätssicherung
Messen/Bewerten

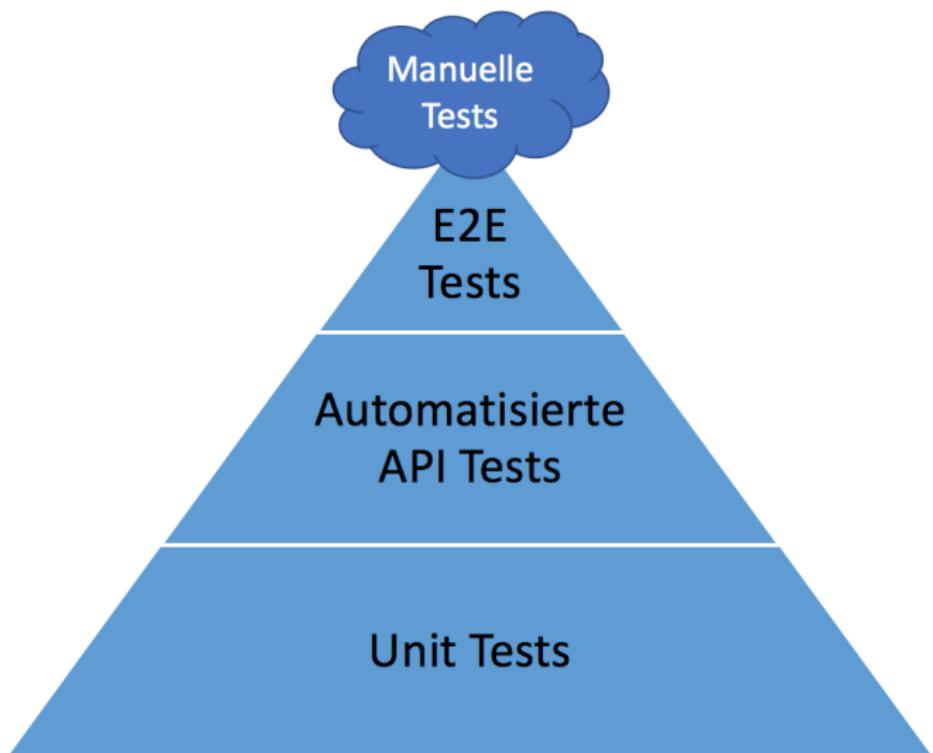
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter





Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

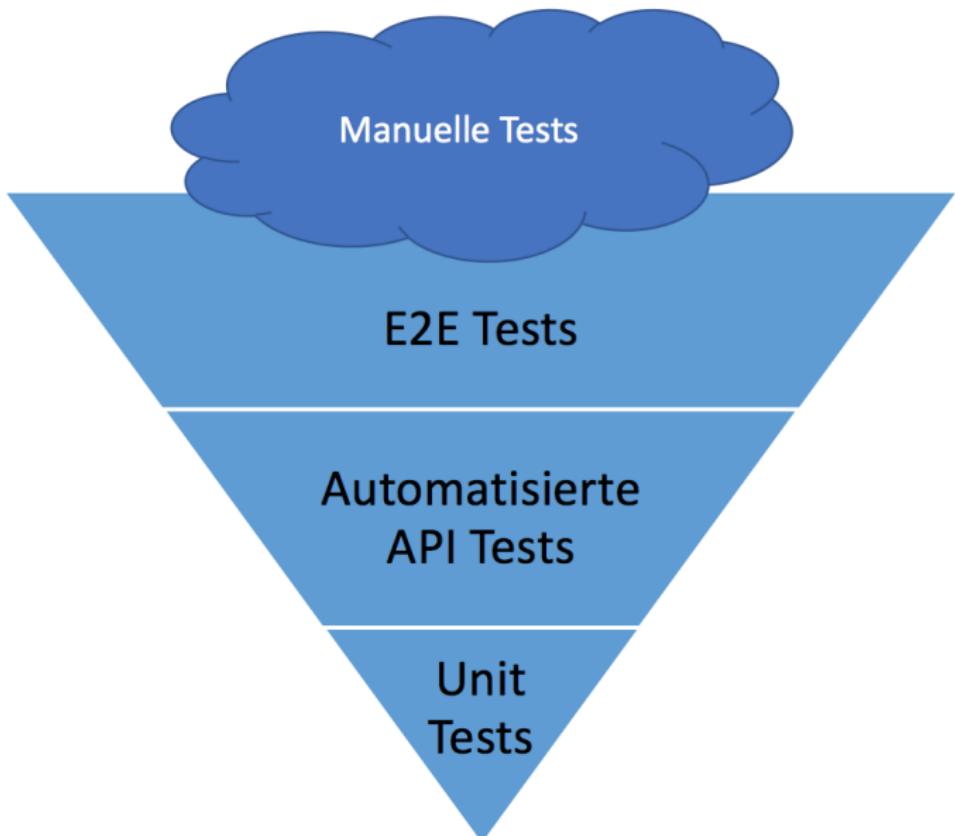
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Weist fehlerfreiheit des Software-Systems nach
- Aufgrund der hohen Komplexität ist Nachweis von Fehlerfreiheit unmöglich
- Ziel des Testens ist es daher möglichst viele vorhandene Fehler zu finden
- Konkretes Programm wird dazu ausgeführt und mit Werten gefüttert
- Ist-Werte werden mit den Soll-Werten aus der Spezifikation verglichen
- Bei Abweichung zwischen Ist-Wert und Soll-Wert liegt ein Fehler vor

Systematische Tests

- Randbedingungen (z.B. Systemumgebung) ist definiert
- Eingaben werden systematisch ausgewählt
- Soll-Werte werden vor dem Test festgelegt
- Gesamter Test wird dokumentiert
- Test und Korrektur finden getrennt statt

Systematische Tests sind stets reproduzierbar und damit objektiv und wiederholbar.

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Es können verschiedene Eigenschaften eines Programms getestet werden woraus sich folgende Testklassifikation ergibt

- Funktionstest
- Last- und Stresstest
- Verfügbarkeitstest
- Installationstest
- Wiederinbetriebnahmetest

Mindestens Funktions- oder Lasttests sollten systematisch durchgeführt werden.

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Ein vollständiger Test deckt alle möglichen Eingaben für ein Programm ab
- Ist ein vollständiger Test fehlerfrei wurde die Korrektheit des Programms bewiesen
- Bereits für einfache Funktion mit ganzahligem Parameter ergibt dies 2^{32} Möglichkeiten
- Aus diesem Grund wird Teilmenge getestet und die Eingaben werden geeignet gewählt

Black-Box Test

Testfälle werden auf Basis der in der Spezifikation geforderten Eigenschaften gewählt. Die innere Beschaffenheit des Programms spielt keine Rolle.

- Testfälle werden aufgestellt sobald die Spezifikation vorliegt
- Folgende Verfahren werden zur Bestimmung der Testfälle betrachtet:
 - Funktionale Äquivalenzklassenbildung
 - Grenzwertanalyse
 - Test spezieller Werte
 - Zufallstest

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Angenommen es existiert eine Menge $E = e_1, e_2, \dots$, von Eingaben auf die das Programm identisch reagiert
- Dann reicht es aus lediglich eine Eingabe dieser Menge zu testen
- Wenn die Eingabe fehlerfrei ist kann man von Korrektheit der restlichen Eingaben ausgehen
- Eine solche Menge wird als Äquivalenzklasse bezeichnet
- Jede Eingabe aus der Äquivalenzklasse ist damit repräsentativ für jede andere Eingabe der Äquivalenzklasse
- In der Praxis wird häufig nur vermutet das Eingaben Äquivalent, dann wird auch von schwacher Äquivalenz gesprochen

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vorgehen bei Äquivalenzklassentest

- 1 Menge der Eingaben wird in Äquivalenzklassen zerlegt
- 2 Testfälle werden aus Eingaben aus möglichst vielen Äquivalenzklassen zusammengestellt
- 3 Ein Testfall sollte höchstens nur eine Eingabe aus einer ungültigen Äquivalenzklasse enthalten um die Fehlerbehandlung klar identifizieren zu können

- Besitzen die Äquivalenzklassen eine natürliche Ordnung werden die Elemente der Klasse gewählt die an den Grenzen liegen
- Annäherung an die Grenzen kann vom gültigen und vom ungültigen Bereich der Klasse erfolgen

- Bei zunehmender Testerfahrung werden Eingaben identifiziert die fehleranfälliger sind als andere
- Diese Eingaben werden für spätere Testfälle dokumentiert
- Beispiele: leere Eingabe, Sonderzeichen

- Testfälle werden zufällig generiert
- Datentypen und Struktur der Eingaben müssen bei Generierung berücksichtigt werden
- Ist ein ergänzendes Verfahren
- Deckt auch nicht naheliegene Eingaben ab

White-Box Test

Der Programmcode ist sichtbar. Es wird ein Strukturtest vorgenommen. Dabei wird die Überdeckung des Codes durch die Testfälle analysiert. Das Ziel ist es eine vorgegebene Überdeckungsrate zu erreichen.

- Um die Überdeckungsrate bestimmen zu können muss das Programm mit zusätzlichen Anweisungen für die Messung instrumentiert werden
- Dadurch kann z.B. gezählt werden wie häufig eine Programmzeile ausgeführt wurde
- Es werden verschiedene Überdeckungskriterien unterschieden
 - Anweisungsüberdeckung
 - Zweigüberdeckung
 - Bedingungsüberdeckung

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Messen und Bewerten

- Während des Projektverlaufs muss der Projektfortschritt durchgehend kontrolliert und gesteuert werden
- Dabei müssen folgende Tätigkeiten ausgeführt werden
 - 1 Das Soll ermitteln
 - 2 Das Ist ermitteln
 - 3 Soll-Ist-Vergleich
 - 4 Bei Abweichung korrigierende Maßnahmen einrichten
- Projekt-Controlling setzt voraus dass bereits geleistete Arbeitsstunden erfasst wurden
- Arbeitsstunden der Mitarbeiter werden jeweiligen Arbeitspaketen zugeordnet
- Ist-Wert kann mit geplanten Soll-Wert (Aufwandsschätzung) verglichen werden

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Wann entstehen Abweichungen zwischen dem Ist-Aufwand
und dem geplanten Soll-Aufwand für ein Arbeitspaket?

- Mangelnde technische Kenntnisse
- Externe Abhängigkeiten die zu Verzögerung führen
- Aufwand für noch zu erledigende Leistungen wird unterschätzt
- Erbrachte Leistung wurde überschätzt
- ...

Fertigstellungsgrad 1

$$FG1 = \text{Ist-Aufwand} / \text{Geschätzter Gesamtaufwand}$$

Selbst wenn die Schätzung des Gesamtaufwands gut ist, wird FG1 am Ende des Projektes ungenau. Daher ist es besser eine transparentere Aufwandsschätzung zu verwenden, die den geschätzten Restaufwand miteinbezieht.

Fertigstellungsgrad 2

$$FG2 = \text{Ist-Aufwand} / (\text{Ist-Aufwand} + \text{Geschätzter Restaufwand})$$

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Ist es aus organisatorischen Gründen nicht möglich den Ist-Aufwand zeitnah zur Verfügung zu stellen kann man den erarbeiteten Wert verwenden.

Erarbeiteter Wert

$$EW = \text{Geplanter Aufwand} - \text{Restaufwand}$$

Daraus ergibt sich der Fertigstellungsgrad 3.

Fertigstellungsgrad 3

$$FG3 = EW / \text{Geplanter Aufwand}$$

Der Fertigstellungsgrad kann auch auf Grundlage der bereits abgeschlossenen Arbeitspakete berechnet werden.

Fertigstellungsgrad 4

$FG4 = \frac{\text{Anzahl abgeschlossener Arbeitspakete}}{\text{Gesamtzahl Arbeitspakete}}$

Bereits begonnene aber noch nicht abgeschlossene Arbeitspakete werden nicht berücksichtigt.

Earned Value Analysis

- Bewertung des Projektfortschritts
- Prognose über den voraussichtlichen Endtermin
- Prognose über zu erwartende Gesamtkosten
- Bereitstellung von Kennzahlen zur Steuerung

BAC	Budget At Completion	Geschätzter Gesamtaufwand des Projekts
PV	Planned Value	Geplante Kosten pro Arbeitspaket
AC	Actual Cost	Ist-Kosten pro Arbeitspaket
EV	Earned Value	Fertigstellungswert

- PV, AC und EV beziehen sich auf den Berichtszeitpunkt
- EV gibt die Ist-Kosten bereits abgeschlossener Arbeit wieder = Wert des Gewerks entsprechend dem erzielten Fortschritt.

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Ein Arbeitspaket ist mit 5 Arbeitstagen geplant. Jeder Arbeitstag kostet 1000 Euro. Zum Berichtszeitpunkt (geplantes Ende des Arbeitspakets, $t=5$) liegt folgende Situation vor: Das AP konnte erst einen Tag später als geplant begonnen werden. Aufgrund von Problemen konnte auch nur die Leistung erbracht werden für die drei Tage vorgesehen waren. Bestimmen Sie PV, AC und EV.

- PV: $1000 * 5 = 5000$
- AC: $1000 * 4 = 4000$
- EV: $5000 * (3 / 5) = 3000$

- Mit folgenden Messgrößen können die Kosten- und Zeitabweichungen von der Planung bestimmt werden.

CV Cost Variance (Kostenabweichung)

$$CV = EV - AC$$

SV Schedule Variance (Zeitabweichung)

$$SV = EV - PV$$

Bestimmen Sie CV und SV für das Arbeitspaket aus Aufgabe 3.6. Was bedeuten diese Werte?

- CV: $3000 - 4000 = -1000$

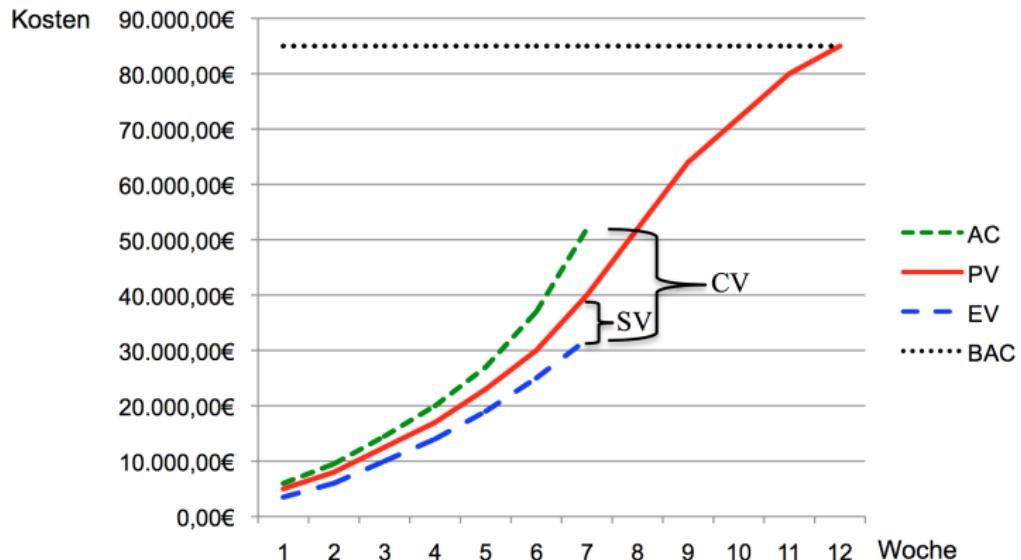
Negative Kostenabweichung deutet auf höhere Kosten im Verhältnis zur Planung hin

- SV: $3000 - 5000 = -2000$

Negative SV deutet auf Zeitverzug hin

Projekt-Controlling - EVA

Mark Keinhörster



Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale
- Prozessmodelle
- Agile Prozessmodelle

Projektmanagement

- Risikomanagement
- Qualitätssicherung
- Messen/Bewerten

Requirements
Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continuous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter

Das Projekt kann über die folgenden Leistungskennzahlen bewertet werden:

$$\begin{array}{lll} \text{CPI} & \text{Cost Performance Index} & \text{CV} = \text{EV} / \text{AC} \\ \text{SPI} & \text{Schedule Performance Index} & \text{SV} = \text{EV} / \text{PV} \end{array}$$

Bei einem guten Projekt sollten beide Werte in einem schmalen und definiertem Intervall um 1 liegen.

CPI >1	Geleistete Arbeit war günstiger als geplant
CPI <1	Geleistete Arbeit war teurer als geplant
SPI >1	Es konnte mehr erreicht werden als geplant
SPI <1	Arbeitsforschritt ist geringer als geplant

Beispiel: Nach 7 Monaten stellen wir bei einem Projekt die folgenden Werte fest:

- AC = 52000
- PV = 40000
- EV = 32000
- SPI = 0.8
- CPI = 0.61

Bis zum Berichtszeitpunkt wurde also 20 Prozent weniger Leistung erbracht als geplant. Da der tatsächliche Aufwand dabei höher war als geplant, liegt das Projekt 39 Prozent unter der Leistung die nach den Kosten zu erwarten war.

Software-Metriken dienen der Quantifizierung der internen Struktur eines Softwaresystems. Sie sind leicht zu messen, müssen jedoch mit Bedacht gewählt und berücksichtigt werden da Sie nicht notwendigerweise Aussagen über die Qualität treffen können.

Dynamische Metriken

Werden während der Ausführung des Programms gesammelt (z.B. während des Testens oder wenn das System produktiv genutzt wird).

Beispiele: Anzahl Bugtickets des letzten Monats, Anzahl offener TCP Connections, Anzahl arbeitender Threads, verwendeter Heapspace, durchschnittliche Round-Trip-Zeit eines Requests ...

Statische Metriken

Werden aus verschiedenen Sichten / Modellen des Systems gesammelt, wie Beispielsweise Klassendiagrammen, dem Quellcode oder der Dokumentation.

Im folgenden befassen wir uns mit statischen Softwaremetriken.

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Abhängigkeit zwischen Klassen
- Kopplung definiert Grad der Abhängigkeit
- Zeigt Einfluss von Änderungen einer Klassen auf andere Klassen auf
- Ziel: Lose/Geringe Kopplung zwischen Klassen

- Grad des Zusammenhangs aller Verantwortlichkeiten, Daten und Methoden einer Klasse
- Ziel ist eine hohe Kohäsion innerhalb einer Klasse/Methode...

Fan-in

- Anzahl der Methoden die eine andere Methode (z. B. `x()`) aufrufen
- Eine hohe Anzahl bedeutet eine starke Kopplung von `x()` zu den restlichen Komponenten
- Änderungen von `x()` kann tiefgreifende Folgen für das Gesamtsystem haben

Fan-out

- Anzahl der Methoden die von Methode `x()` aufgerufen werden
- Eine hohe Anzahl bedeutet eine hohe Komplexität von `x()` da viel Steuerungslogik zur Koordination der Methodenaufrufe nötig ist

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Misst die Größe des Programms anhand des Quellcodes
- Allgemein gilt je größer eine Komponente ist desto komplexer und fehleranfälliger ist sie
- Eine der zuverlässigsten Metriken zur Vorhersage fehleranfälliger Komponenten

- Anzahl linear unabhängiger Pfade innerhalb einer Methode
- Gibt die minimale Anzahl an Tests vor die benötigt werden um eine komplette Testabdeckung der Methode zu erzielen
- Mögliche Berechnung: Anzahl der Verzweigungen mit genau 2 Zweigen + 1
- Bei verschachtelten Verzweigungen oder Verzweigungen mit mehreren möglichen Wegen werden diese auf Binärverzweigungen heruntergebrochen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Durchschnittliche Länge von Variablen-, Methoden-, Klassennamen ...
- Je länger die Namen desto höher ist oftmals die Ausdrucksstärke und Verständlichkeit des Programms

- Misst die Tiefe der Verzweigungen in einer Methode
- Sehr tiefe Verzweigungsbäume sind häufig schwer zu verstehen, fehleranfällig und lassen sich nur mit Seiteneffekten modifizieren

- Anzahl der Methoden einer Klasse Gewichtet nach Komplexität
- Einfache Methoden haben eine Komplexität von 1
- Komplexe Methoden können beliebig hohe Gewichtungen erhalten
- Je höher der Wert der Metrik desto komplexer ist die Gesamtstruktur der Klasse
- Oftmals sind komplexe Klassenstrukturen gering Kohäsiv und schwer wiederverwendbar

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Anzahl der Stufen in einem Vererbungsbaum in dem eine Kindklasse die Attribute und Operationen der Vaterklasse erbt
- Je tiefer der Baum desto komplexer die Komponente
- Das Nutzen der Blätter (tiefsten Kindklassen) setzt das Verständnis einer hohen Anzahl von Vaterklassen voraus

Number of children (NOC)

- Misst die Anzahl der direkten Kindklassen einer Vaterklasse
- NOC misst die Breite einer Klassenhierarchie während DIT die Tiefe misst
- Ein hoher Wert gilt als Indikator für hohe Wiederverwendung

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Requirements Engineering

- Anforderungen des Kunden an die Software sind der zentrale Faktor in einem Softwareprojekt!
- Wenn die Anforderungen des Kunden nicht erfüllt werden ist die Software wertlos
- Die systematische Ermittlung, Analyse und Spezifikation der Anforderungen wird als Systemanalyse oder Requirements Engineering bezeichnet

Anforderung

Anforderungen (requirements) legen fest, was von einem Softwaresystem als Eigenschaft erwartet wird.

Komplexität der Funktionen

- Software wird komplexer je mehr Funktionen sie besitzt
- Beispiel: Integrierte Büroanwendungen
 - Tabellenkalkulation, Präsentationen, Textverarbeitung, Datenbank
 - 1000 - 1500 Funktionen

Komplexität der Daten

- Wird durch Vielzahl von Datenstrukturen oder sehr komplexen Datenstrukturen definiert
- Beispiel: DB-orientierte Systeme, Datawarehouses, Big Data Applikationen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Komplexität der Algorithmen

- Systeme die komplexe numerische Berechnungen durchführen

Komplexität des zeitabhängigen Verhaltens

- Wird durch viele nebenläufige Prozesse, gegenseitiges Ausschließen oder Zeitbedingungen gekennzeichnet
- Beispiele sind Betriebssysteme, Prozesssteuerungen, verteilte Systeme

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Komplexität der Systemumgebung

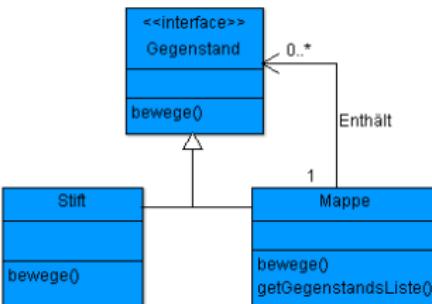
- Systeme bei denen es im wesentlichen auf das Zusammenwirken mit dem Gesamtsystem ankommt
- Beispiel: Embedded Systems (Flugzeugsteuerungen, Fahrzeugsysteme), IOT Lösungen

Komplexität der Benutzeroberfläche

- Systeme mit komplexer Interaktion zwischen Anwender und Computersystem
- Beispiel: CAD-Systeme, CASE-Systeme, Büro- und Enterpriseanwendungen

Das Modellierungskonzept ist entsprechend der Komplexität des zu entwickelnden Produkts auszuwählen. Weist das zugrundeliegende Produkt beispielsweise eine hohe Komplexität der Funktionen auf, so muss das verwendete Konzept es erlauben eine Vielzahl von Funktionen zu modellieren.

- Unified Modeling Language
- Sprache zur objektorientierten Modellierung
- Standardisiert von der OMG
- Auch zur Implementierung genutzt



Berücksichtigung aktueller Trends der Softwareentwicklung

- Patterns
- Komponentenorientiertes Design
- Anwendungsfälle modellieren
- Ausführbarkeit von Modellen

OOA (Objektorientierte Analyse) = Anwendersicht

- Modell der Anforderungen

OOD (Objektorientiertes Design) = Entwicklersicht

- Details und Randbedingungen

OOP (Objektorientierte Progr.) = Entwicklersicht

- Implementierung

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

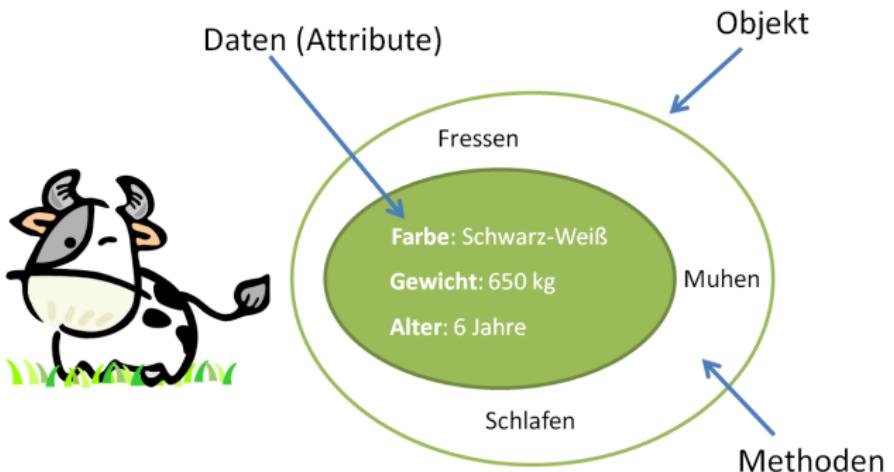
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

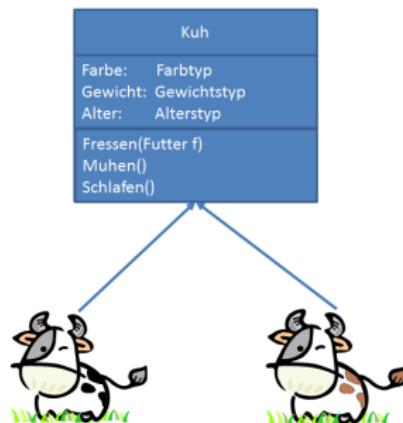
Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Exkurs OOP: Klassen als Objektbauplan

- Klasse fungiert als Bauplan für Objekte
- Aus diesem Bauplan lassen sich beliebig viele Objekte erzeugen
- Objekte aus einer Klasse besitzen die gleichen Methoden
- Objekte aus einer Klasse besitzen die gleichen Attribute (Werte können jedoch variieren)
- Klassen können durch Vererbung weiter spezialisiert werden



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

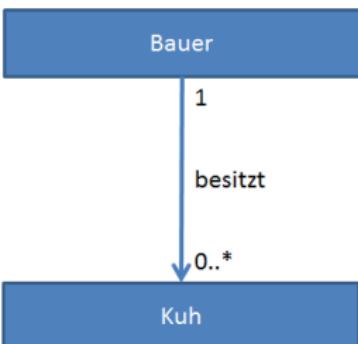
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

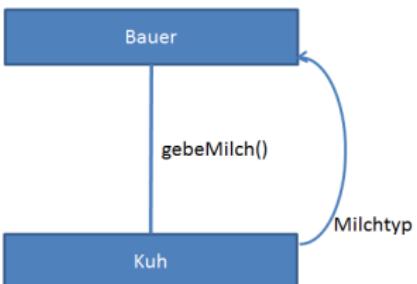
Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Zwischen Objekten können Beziehungen existieren
- Beziehung zwischen Bauer und Kuh:
 - 1 Der Bauer kann 0 bis n Kühe besitzen
 - 2 Eine Kuh gehört genau einem Bauer
 - 3 Keine Kuh kann einen Bauer besitzen (gerichtete Beziehung)



Exkurs OOP: Methodenaufrufe

- Objekte können sich untereinander Botschaften senden
- Damit wird der Empfänger aufgefordert etwas auszuführen
- Erweitern wir die Klasse "Kuh"
 - 1 Kuh bekommt die Methode: "gebeMilch"
 - 2 "gebeMilch" gibt ein Objekt vom Typ "Milchtyp" zurück
- Der Bauer löst durch senden einer Botschaft die Methode "gebeMilch" aus
- Die Kuh führt die Methode aus und gibt Milchtyp-Objekt zurück

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale
- Prozessmodelle
- Agile Prozessmodelle

Projektmanagement

- Risikomanagement
- Qualitätssicherung
- Messen/Bewerten

Requirements
Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continuous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter

Use Case

Ein Use Case beschreibt Szenarien einzelner Interaktionen zwischen dem Nutzer und dem System.

Zielsetzung

- Spezifikation der Funktionalität des Systems
- Identifikation: Wer / was arbeitet mit dem System
- Trennung zwischen dem was das System leistet und was außerhalb bearbeitet wird
- Zergliederung in Einheiten mit Akteuren und den zugehörigen Anwendungsfällen

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Use Case Beispiel

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

- Basismodelle
- Monumentale Prozessmodelle
- Agile Prozessmodelle

Projektmanagement

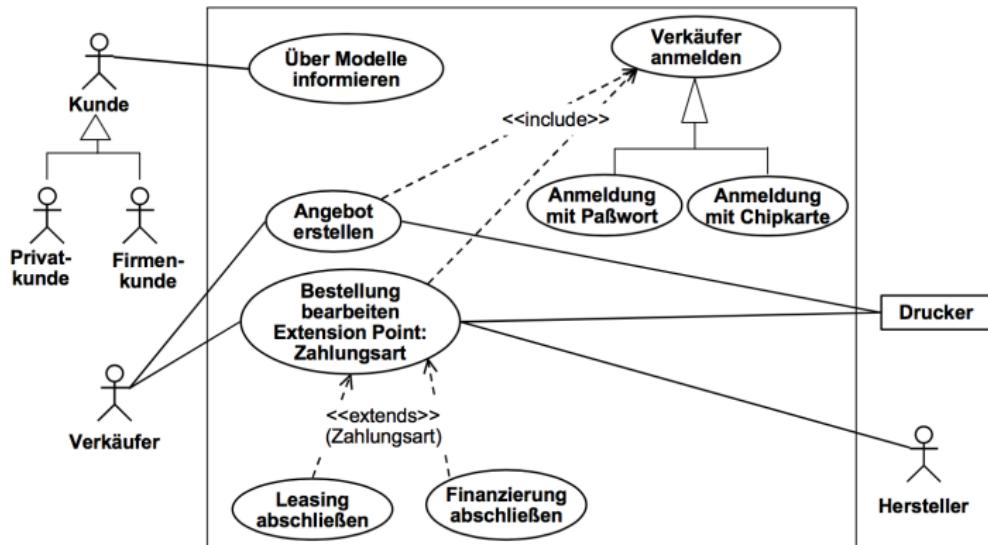
- Risikomanagement
- Qualitätssicherung
- Messen/Bewerten

Requirements
Engineering

- Use Cases
- Klassendiagramme

Exkurs

- Continuous Integration / Delivery mit Docker
- Data Science mit Apache Spark und Jupyter



Beschreiben die Interaktionen zwischen den Akteuren und dem System

- Wie wird das System verwendet?
- Welches Verhalten zeigt das System nach außen?
- Use Cases werden stets von einem Akteur angestoßen, um bestimmte Funktionalität des Systems zu nutzen

Beschreiben alle Abläufe die zur Durchführung nötig sind

- Nur Abläufe und Schritte die nach außen sichtbar sind
- Keine internen Strukturen

Sind in sich abgeschlossen

- Alle Abläufe sind vollständig enthalten
- Jeder Use Case kann für sich allein bestehen und ist für sich allein sinnvoll
- Jeder Use Case liefert ein Resultat für mindestens einen Akteur

Alle Use Cases modellieren in ihrer Gesamtheit das funktionale Verhalten des Systems

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Ein Akteur ist jemand/etwas, der/das
 - ... mit dem zu entwickelnden System zusammenarbeitet
 - ... eine Interaktion ausführt
- Ist stets außerhalb des Systems
- Sind Abstraktionen (Personen/Rollen, Maschinen, Systeme)
- Legen die Systemgrenzen fest



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle

Monumentale

Prozessmodelle

Agile Prozessmodelle

Projektmanagement

Risikomanagement

Qualitätssicherung

Messen/Bewerten

Requirements
Engineering

Use Cases

Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

Data Science mit

Apache Spark und

Jupyter



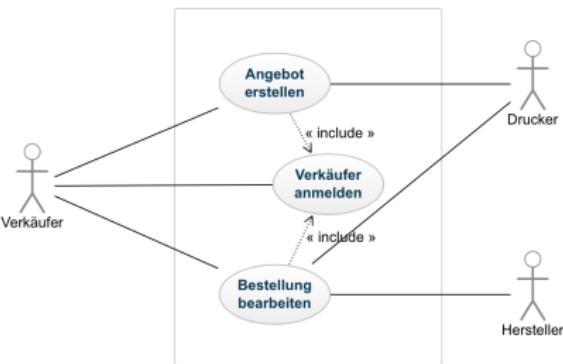
- Aktivitäten die mit dem System durchgeführt werden
- Aktivitäten die nach außen hin sichtbar sind



- Zusammenhänge von Akteuren und ihren Use Cases

Fortgeschrittene Konzepte - Include

- Mehrere Use Cases besitzen gleiche Teile
- Gemeinsamer Teil wird in eigenen Use Case ausgelagert
- Wird bei ausführung an entsprechender Stelle aufgerufen
- Ausgelagerter Teil muss sinnvolle Einheit ergeben
- Gerichtet vom Basisfall zum gemeinsamen Teil



Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

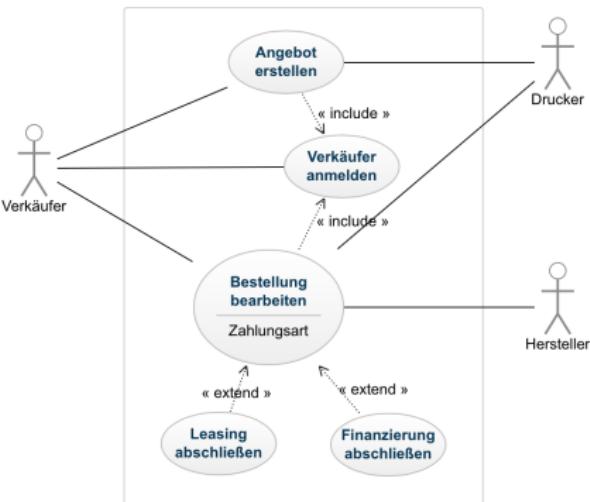
Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Fortgeschrittene Konzepte - Extend

- Use Case kann an definierten Punkten erweitert werden
- Erweiterung trennt definiertes Verhalten von Optionalem / Ergänzendem
- Basis Use Case ist auch ohne Erweiterung sinnvoll
- Gerichtet von Optionalem zu Basis Use Case



Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

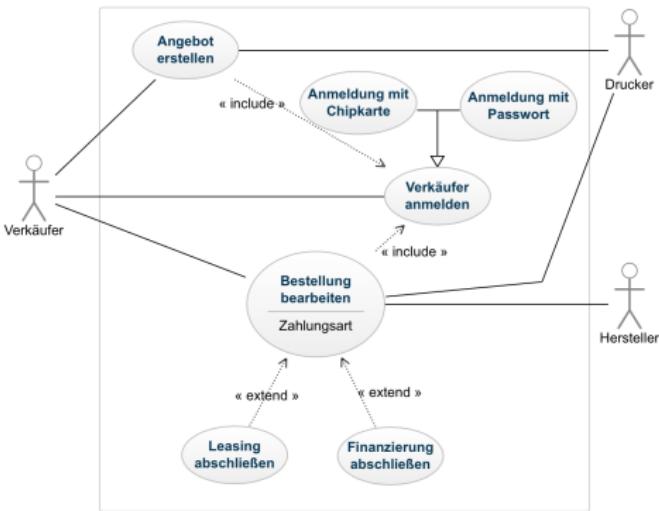
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Möglich bei semantisch ähnlichen Funktionalitäten
- Unter-Use Case erbt alle Eigenschaften des Ober-Use Case
- Unter-Use Case kann modifiziert oder ergänzt werden
- Unter-Use Case kann an jeder Stelle anstatt des Ober-Use Case verwendet werden



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

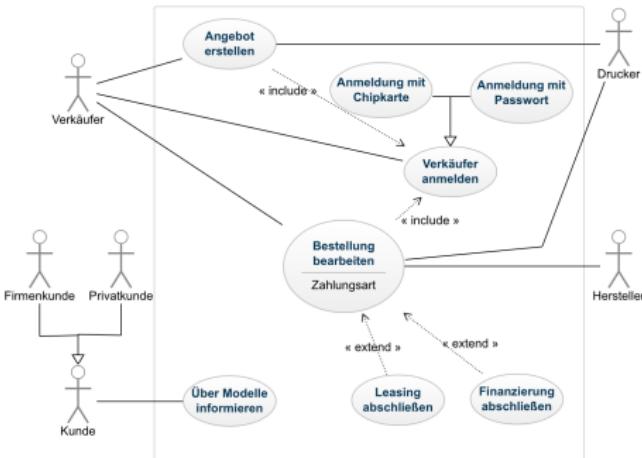
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Unter-Akteur erbt alle Use Cases des Ober-Akteurs
- Unter-Akteur kann eigene Use Cases erhalten



Verständlichkeit

Use Cases dienen der Kommunikation mit den Anwendern.

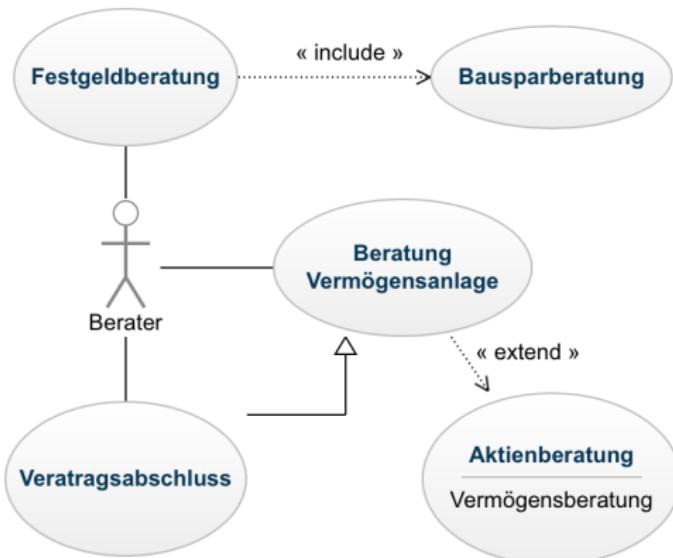
Daher müssen sie von diesen verstanden werden können. Es ist nicht Ziel alle der zur Verfügung stehenden Konzepte zu verwenden, denn damit entsteht eine höhere Komplexität und somit erschwert Lesbarkeit / Verständlichkeit.

Use Case
Zusammenfassung
Akteure
Vorbedingungen
Ablaufbeschreibung
Verwendung
Erweiterung
Alternativen
Nachbedingungen
Fehlschlag
Sonstiges

Name des Use Case
Kurzbeschreibung
Akteure die den Use Case auslösen oder involviert sind
Voraussetzungen die zur Ausführung erfüllt sein müssen
Strukturierte Ablaufbeschreibung (alle Standardfälle)
Auflistung aller Include-Beziehungen
Auflistung aller Extend-Beziehungen
Optionen und Alternative Ausführungen
Resultate die der Use Case liefert
Erwarteter Zustand wenn das Ziel nicht erreicht wird
Nutzen des Use Case; Probleme die der Use Case löst

Übung 4.1

Untersuchen Sie dieses Use Case Diagramm auf Fehler.
Berichtigen Sie das Diagramm derart, dass es einen korrekten Sachverhalt wiedergibt.



Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Übung 4.1 - Lösung

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

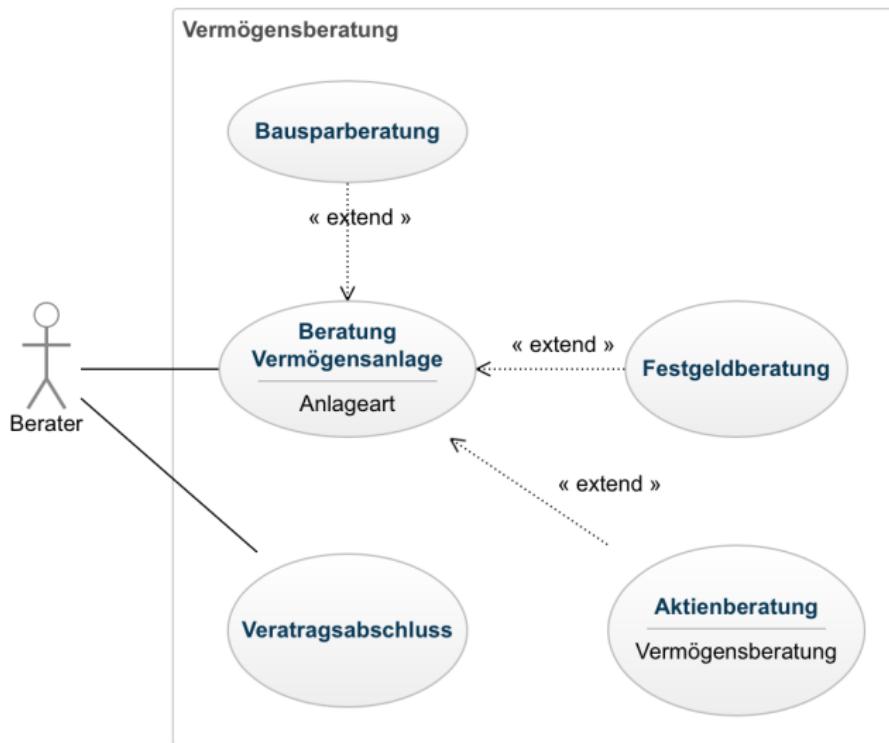
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

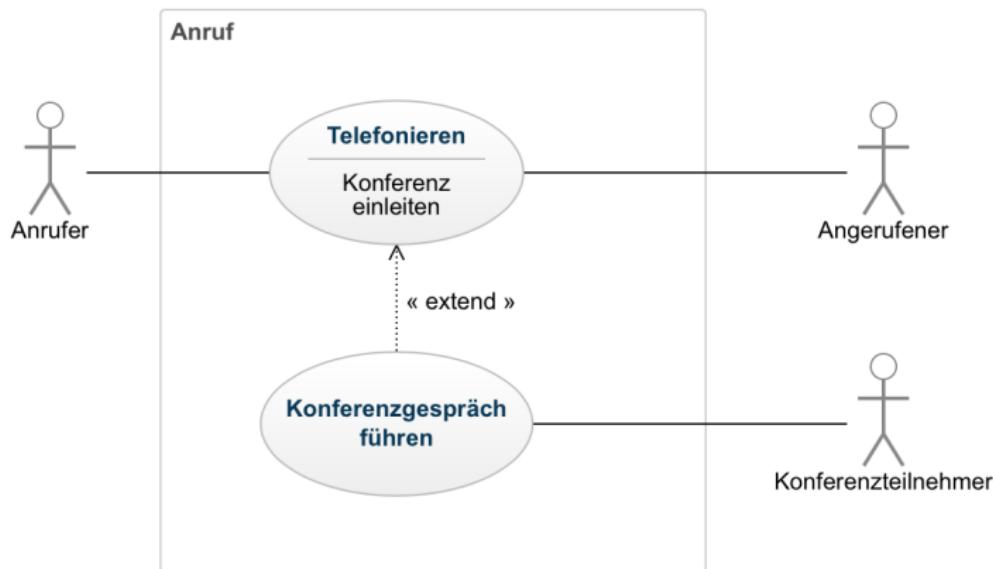
Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Erstellen Sie ein Use Case Diagramm für das Anrufen (das herkömmliche Telefongespräch zwischen 2 Personen). Berücksichtigen Sie dabei die Möglichkeit einer Konferenzschaltung (Hinzufügen eines weiteren Gesprächspartners zum bestehenden Telefongespräch). Beschreiben Sie die einzelnen Use Cases aus Ihrem Diagramm. Dazu verwenden Sie das in der Vorlesung vorgestellte Template.

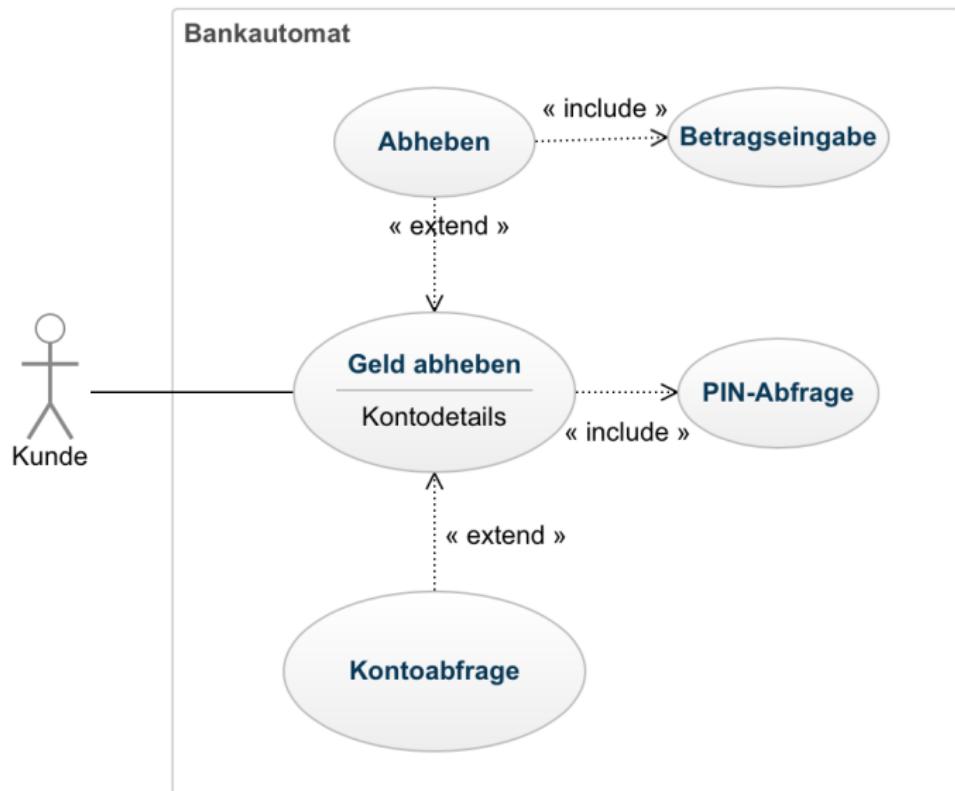


Use Case Diagramm lässt sich beliebig erweitern)

Beschreiben Sie die Benutzung eines Bankautomaten mit dem Ziel Geld abzuheben in Form eines Use Cases.

Beachten Sie dabei, dass Use Cases nur Abläufe / Schritte enthalten, die nach außen sichtbar sind; sie enthalten keine internen Strukturen.

Beschreiben Sie die einzelnen Use Cases aus Ihrem Diagramm. Dazu verwenden Sie das in der Vorlesung vorgestellte Template.



Beschreiben Sie die Verwaltung einer Bibliothek bei der Buchausleihe in Form eines Use Cases.

Beschreiben Sie die einzelnen Use Cases aus Ihrem Diagramm. Dazu verwenden Sie das in der Vorlesung vorgestellte Template.

Übung 4.4 - Lösung

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

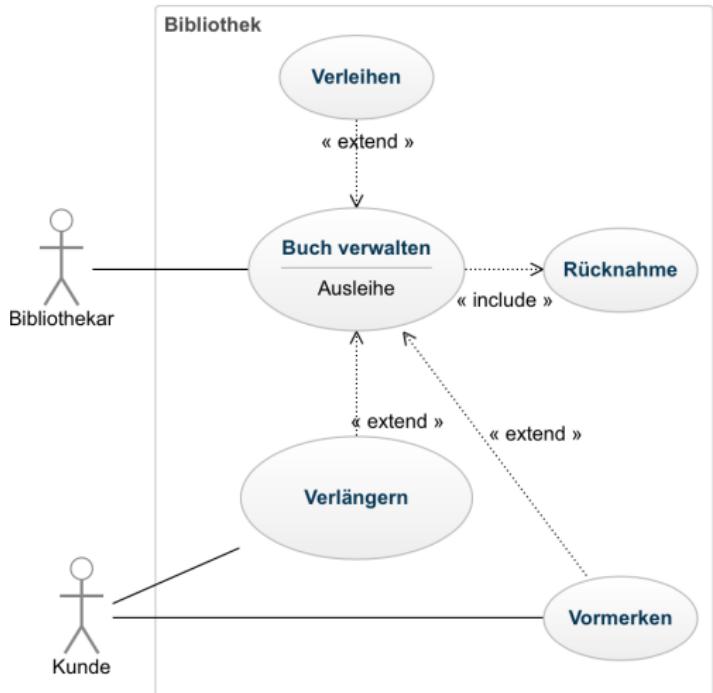
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Vorteile

- Leichter Zugang zum System
- Besseres Verständnis des Systems (Abgrenzung innerhalb und außerhalb des Systems)
- Basis für Klassendefinition
- Erleichtert Kommunikation zwischen Entwickler und Anwender
- Basis für System- und Abnahmetests
- Grundlage für Benutzerdokumentation

Nachteile

- Gefahr der funktionalen Zerlegung (zu viel kleine Use Cases)
- Kontrollstrukturen lassen sich nicht einfach darstellen (nur in Beschreibung)
- User Interface / User Experience lässt sich nicht abbilden

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Als Rechteck dargestellt
- In beliebig viele Compartments unterteilt
- Inhalt der Compartments frei wählbar
- Verschiedene Sichten durch Verzicht auf Compartments möglich

Konvention:

Klasse
 Attribut1: Integer
 Attribut2: String
 operation1(String, Boolean): Integer

Syntax

Sichtbarkeit / Name : Typ Multiplizität = Anfangswert

{Einschränkungen}

(Mit Ausnahme des Namens sind alle Angaben optional)

- Beschreiben Eigenschaften einer Klasse
- Zeigen Daten die von Objekten angenommen werden können
- Jedes Attribut hat einen Namen (Substantiv, bsp. Farbe, Preis)
- Notation Klassenattribut: Klassenattribut
- Abgeleitetes Attribut wird aus anderen Attributswerten berechnet, Notation: /Attribut

Software
Engineering

Mark Keinhrörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- **public:** +
- **private:** -
- **package:** ~
- **protected:** #

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Die Multiplizität legt die Unter- und Obergrenze der Anzahl der unter einem Attributnamen ablegbaren Instanzen fest.

- 0..1 optionales Attribut (höchstens ein Wert)
- 1..1 zwingend, genau ein Wert
- 0..* optional beliebig
- 1..* mindestens einer, belieb viele
- n..m mindestens n, höchstens m

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- readOnly: Attribute deren Wert nicht verändert werden darf (Konstanten)
- ordered: Legt Reihenfolge der Inhalte eines Attributs fest
- unique: Legt fest ob Attribut duplikatsfrei ist
- In der UML Spezifikation sind weitere Eigenschaftswerte enthalten

Beispiel Attribute

Beispiel



Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Übung 4.5

Die Spezifikation ist fehlerhaft, identifizieren Sie die Fehler.

Name des Attributs	Name	Gehalt
Beschreibung	Gibt den Nachnamen an	Gibt das Gehalt an
Typ	String	Float
Anfangswert	Meier	0
Restriktion	Keine Doppelnamen	$1000 < \text{Gehalt} < 4000$
Klassenattribut	nein	ja
Abgeleitetes Attribut	ja	nein
Muss-Attribut	nein	nein
Attributwert nach dem ersten Eingeben nicht mehr Änderbar	ja	ja

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Übung 4.5 - Lösung

Name des Attributs	Name	Gehalt
Beschreibung	Gibt den Nachnamen an	Gibt das Gehalt an
Typ	String	Float
Anfangswert	Meier (F)	0
Restriktion	Keine Doppelnamen	1000 < Gehalt < 4000
Klassenattribut	nein	ja (F)
Abgeleitetes Attribut	ja (F)	nein
Muss-Attribut	ja	nein (F)
Attributwert nach dem ersten Eingeben nicht mehr Änderbar	ja (F)	ja (F)

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Syntax

Sichtbarkeit Name (Parameter) : Rückgabetyp
{Einschränkungen}

- Ausführbare Tätigkeit im Sinne eines Algorithmus
- Es muss darauf geachtet werden, welcher Klasse Operationen sinnvoll zugeordnet werden.
Klasse Auto sollte die Operation "ausstattungHinzufügen" zugeordnet haben und nicht die Klasse Ausstattung.
- Notation Klassenoperation: operation(), auch Konstruktoren
- Parameter kann Richtung haben (in, out, inout)

Software
Engineering

Mark Keinhörster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Beispiel

- +halloWelt()
- +add(summand1, summand2):int
- -div(divident:int, divisor:int):double
- #sub(in minuend:double, in subtrahend:double, out resultat:double)
- inc(inOut wert)
- setLstKI(lohnsteuerKlasse:int=1)

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Definition

Eine Assoziation drückt die Möglichkeit aus Instanzen einer Klasse mit Instanzen derselben oder anderer Klassen zu verbinden.

- Bennung ist optional, kurze prägnante Verben
- Bezeichnung von Rollen der Klassen in einer Assoziation ist optional, Verwendung von Substantiven

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

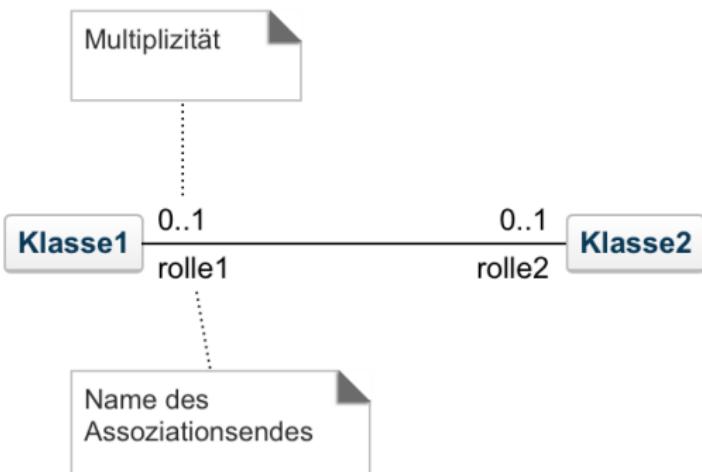
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

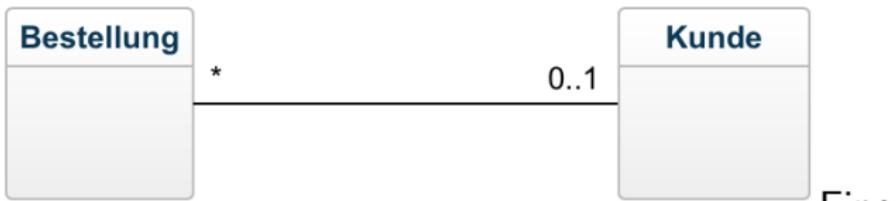
Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Beispiel

Beispiel



Kann-Assoziation hat als Untergrenze die Multiplizität 0.
Eine Muss-Assoziation hat die Multiplizität größer 1 oder
eine Untergrenze größer 1

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

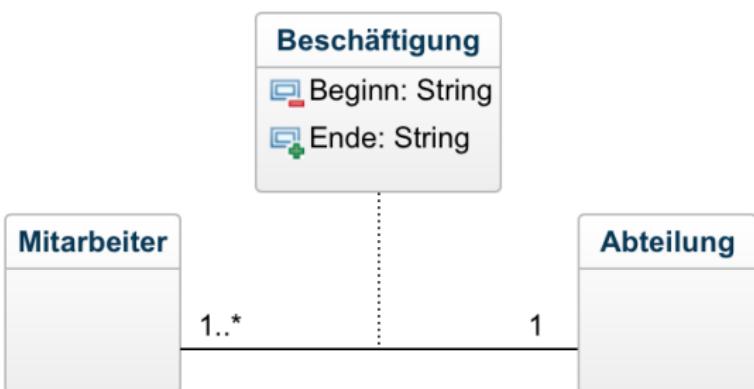
Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

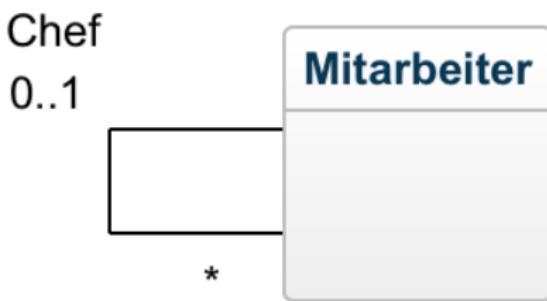
Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter





Eine reflexive Assoziation besteht zwischen Objekten derselben Klasse



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

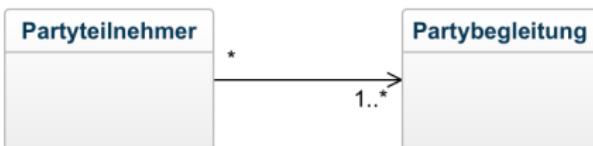
Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Die folgende Assoziation definiert eine Beziehung ohne Angaben zu ihrer Navigierbarkeit



Instanzen der Klasse Partyteilnehmer kennen Instanzen der Klasse Partybegleitung. Die potenzielle Partybegleitung kennt jedoch nicht den Partyteilnehmer.

Die bidirektionale Navigierbarkeit weist entsprechend Pfeile an beiden Assoziationsenden auf.



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

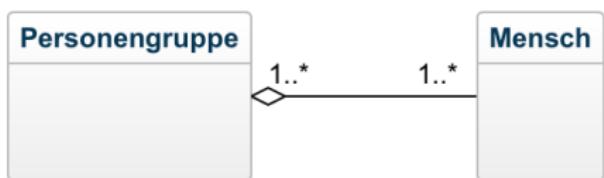
Requirements
Engineering

Use Cases
Klassendiagramme

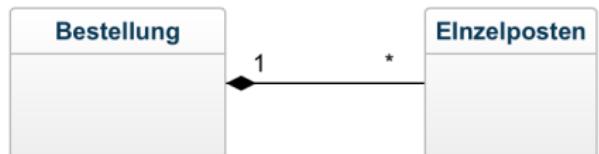
Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Aggregation ist eine Teil-Ganzes-Beziehung, die sich durch "besteht aus" - für das Assoziationsende des Ganzen - und "ist Teil von" für das Ende der Teile - beschreiben lässt.



- Komposition ist eine starke Aggregation
- Die Lebensdauer der Teile ist in diejenige des Ganzen eingebettet



Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

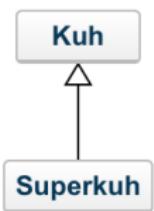
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Vererbung bedeutet, dass eine spezialisierte Klasse über die Eigenschaften, das Verhalten und die Assoziationen einer oder mehrerer allgemeiner Klassen verfügen kann.



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

In einer Schule soll der Lehrbetrieb rechnergestützt verwaltet werden:

- 1 Jeder Lehrer kann bis zu vier Fächer unterrichten
- 2 Eine Klasse wird von verschiedenen Lehrern in unterschiedlichen Fächern unterrichtet
- 3 Jeder Klasse ist ein bestimmter Lehrer als Klassenlehrer zugeordnet
- 4 Jeder Lehrer darf nur für eine Klasse Klassenlehrer sein
- 5 Jede Unterrichtsstunde findet in einem bestimmten Raum zu einer bestimmten Zeit statt und wird von einem Lehrer vor einer Klasse abgehalten
- 6 Jede Klasse hat zwischen 30 und 35 Unterrichtsstunden
- 7 Es gibt Fächer die nicht immer unterrichtet werden können

Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitätssicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

Exkurs

Vagrant

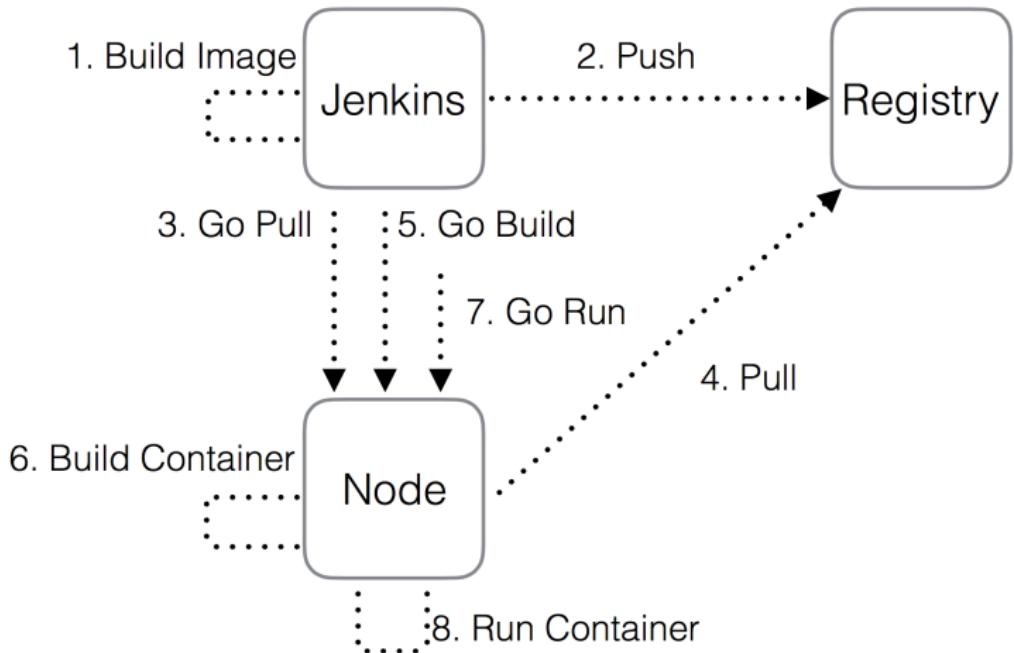
Erlaubt einfache und reproduzierbare Erstellung von Entwicklungsumgebungen. Provisionierung auf Basis von Virtual Box, VMWare, AWS ... Nutzung von Tools wie Chef, Puppet, Shell-Skripte

Docker

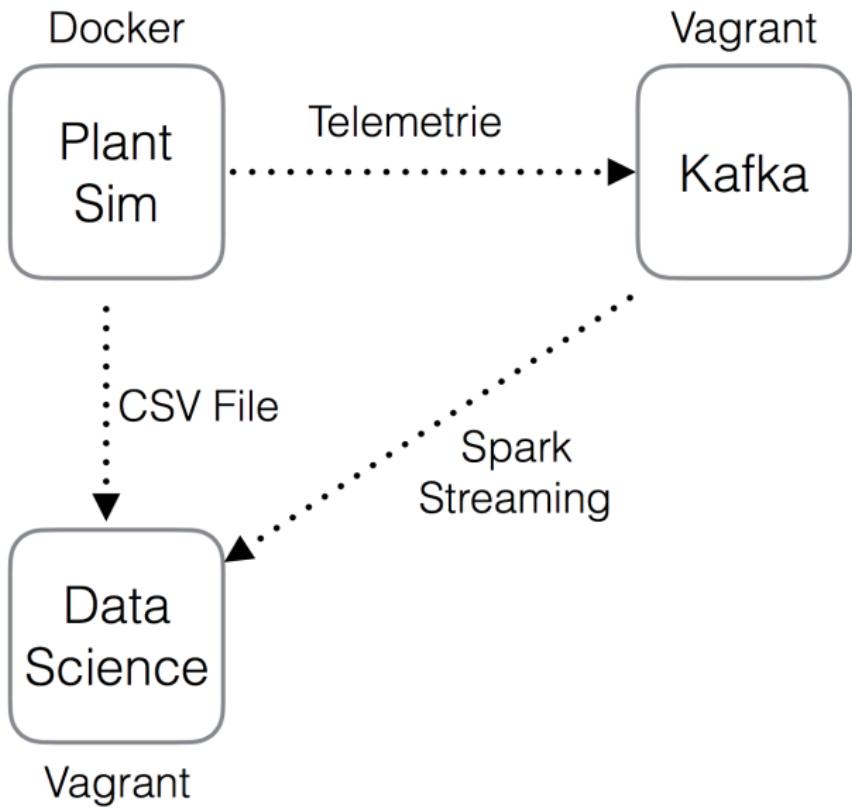
Docker ist eine Plattform zur Virtualisierung in Containern.

- Container sind Software auf Filesystem
- Nutzen Kernel des Hosts
- Sind Prozesse auf Host

Continuous Integration / Delivery mit Docker



Data Science mit Apache Spark und Jupyter



Software
Engineering

Mark Keinhorster

Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

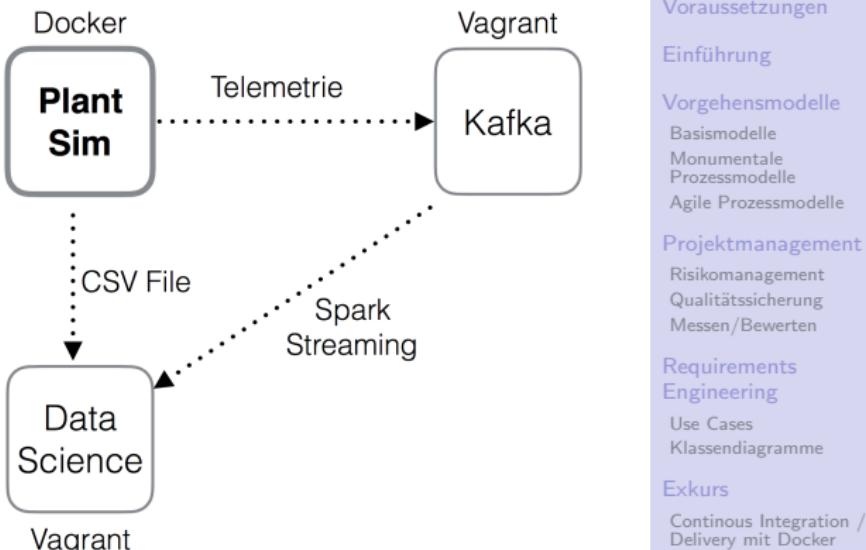
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter

- Python für "einfaches Hacken"
- Von außen steuerbar
- Plattformunabhängig
- In Entwicklungsprozess integriert
- Der Daten-Generator



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

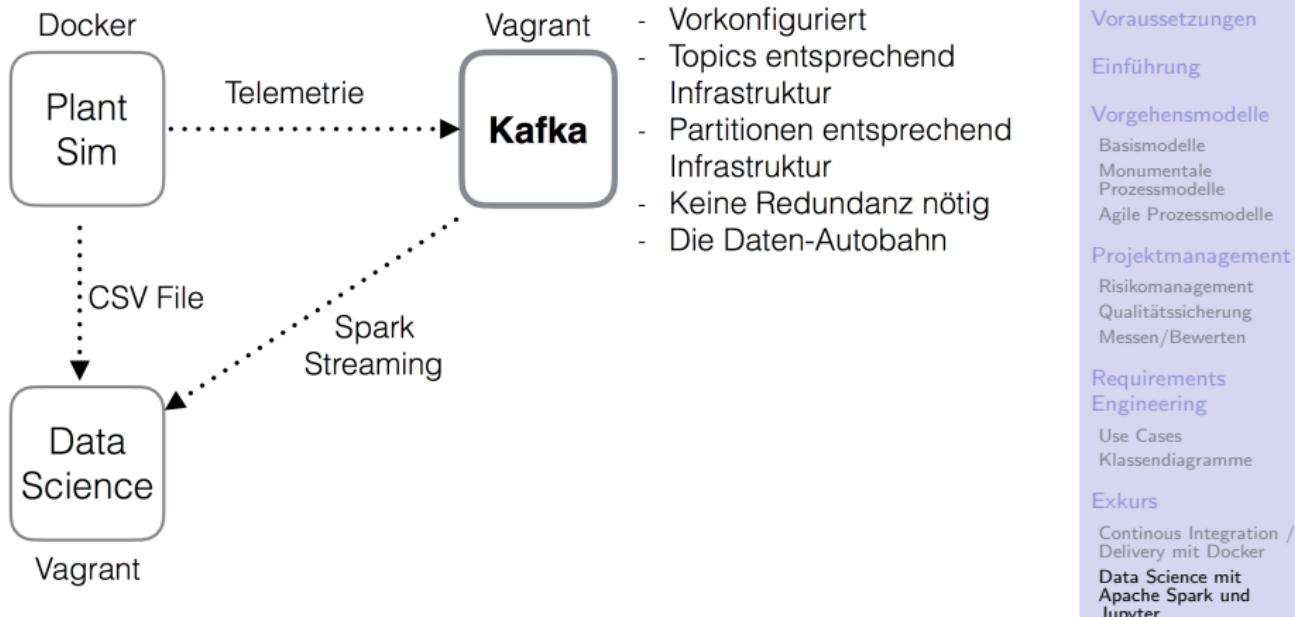
Risikomanagement
Qualitäts sicherung
Messen/Bewerten

Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker
Data Science mit
Apache Spark und
Jupyter



Voraussetzungen

Einführung

Vorgehensmodelle

Basismodelle
Monumentale
Prozessmodelle
Agile Prozessmodelle

Projektmanagement

Risikomanagement
Qualitäts sicherung
Messen/Bewerten

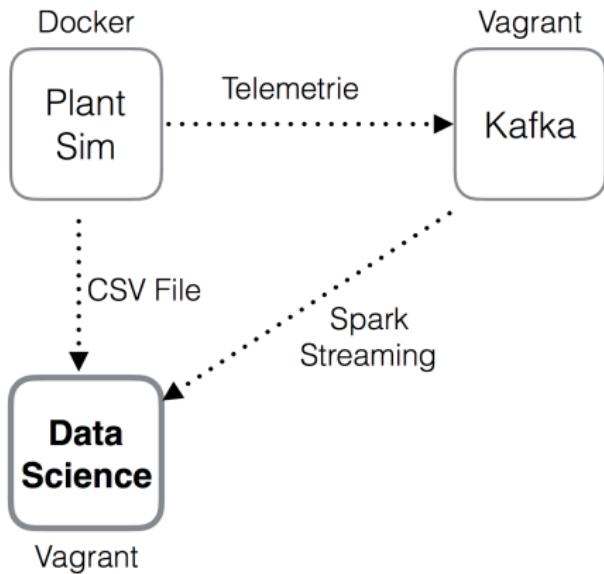
Requirements
Engineering

Use Cases
Klassendiagramme

Exkurs

Continuous Integration /
Delivery mit Docker

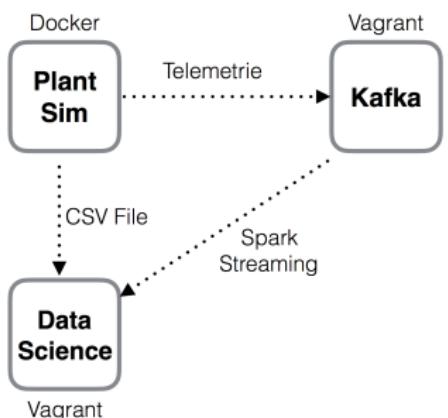
Data Science mit
Apache Spark und
Jupyter



- Großes Repertoire an Tools
- Apache Spark entsprechend Infrastruktur
- Jupyter Notebooks
- In Entwicklungsprozess integriert
- Der Daten-Transformator

- Ein wenig Development für den Fachbereich
- Immer auf dem neuesten Stand
- Direktes Feedback in Form von Code
- Business "nah" am Code

- Analysten werden zu Data-Scientists
- Vorgefertigter "Glue Code"
- Jupyter Notebooks können von allen genutzt werden
- Direktes Feedback in Form von Code



- Kann Blackbox sein
- Insights für Developer
- Entspricht einer Integrationsumgebung