Software Engineering

Mark Keinhörster

FOM
Hochschule für Ökonomie und Management

3. März 2017

Inhaltsverzeichnis

Voraussetzungen

Ihre Erwartungen an die Veranstaltung

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

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

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

Softwarekatastrophe ARIANE 5

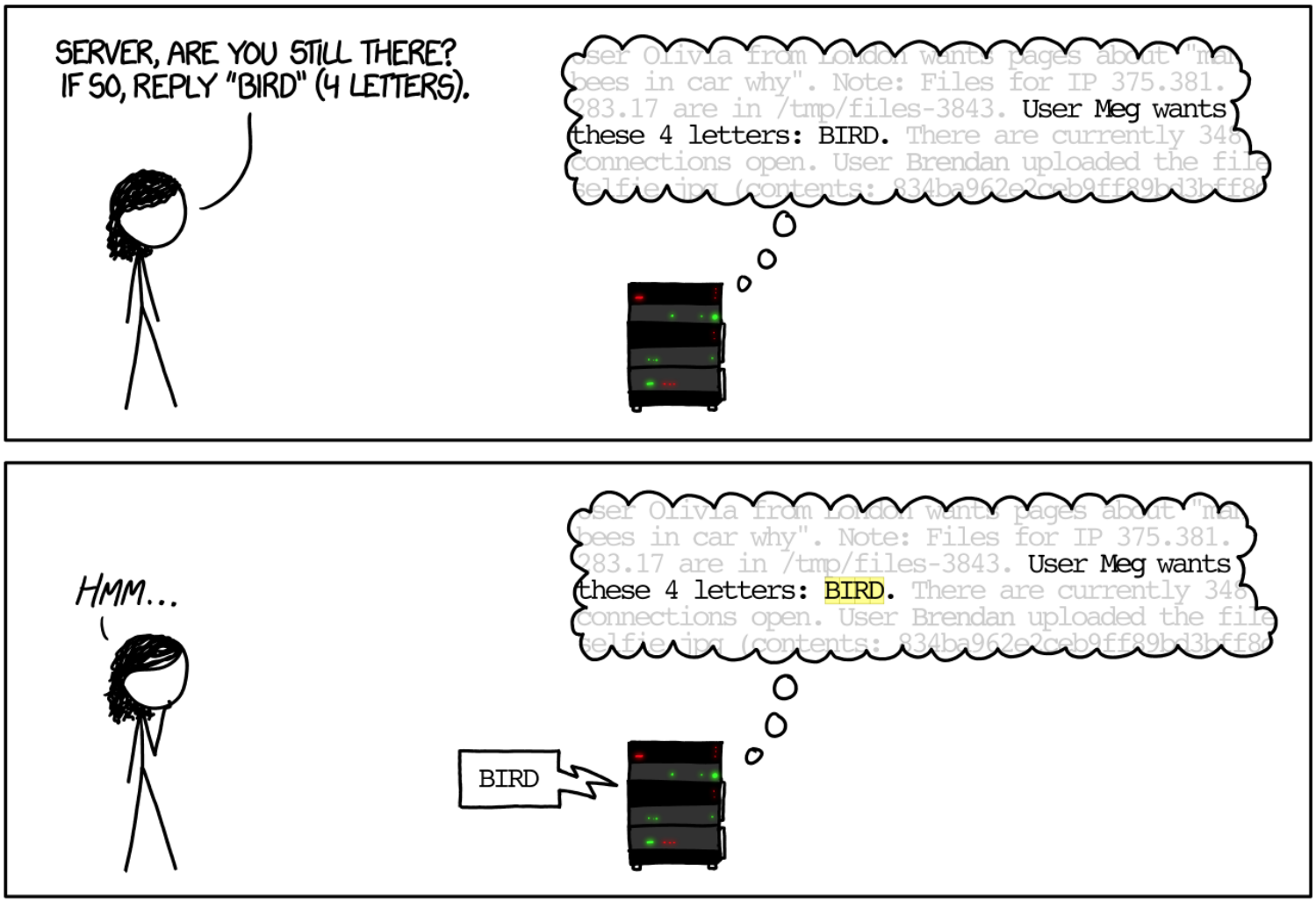
On June 4, 1996, on its maiden flight, the Ariane-5 was launched and performed perfectly for approximately 40 seconds. Then it began to veer off course. At the direction of the Ariane ground controller, the rocket was destroyed by remote control. . . . total cost of the disaster was 500 million dollar.

- 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.

Softwarekatastrophe Heartbleed

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

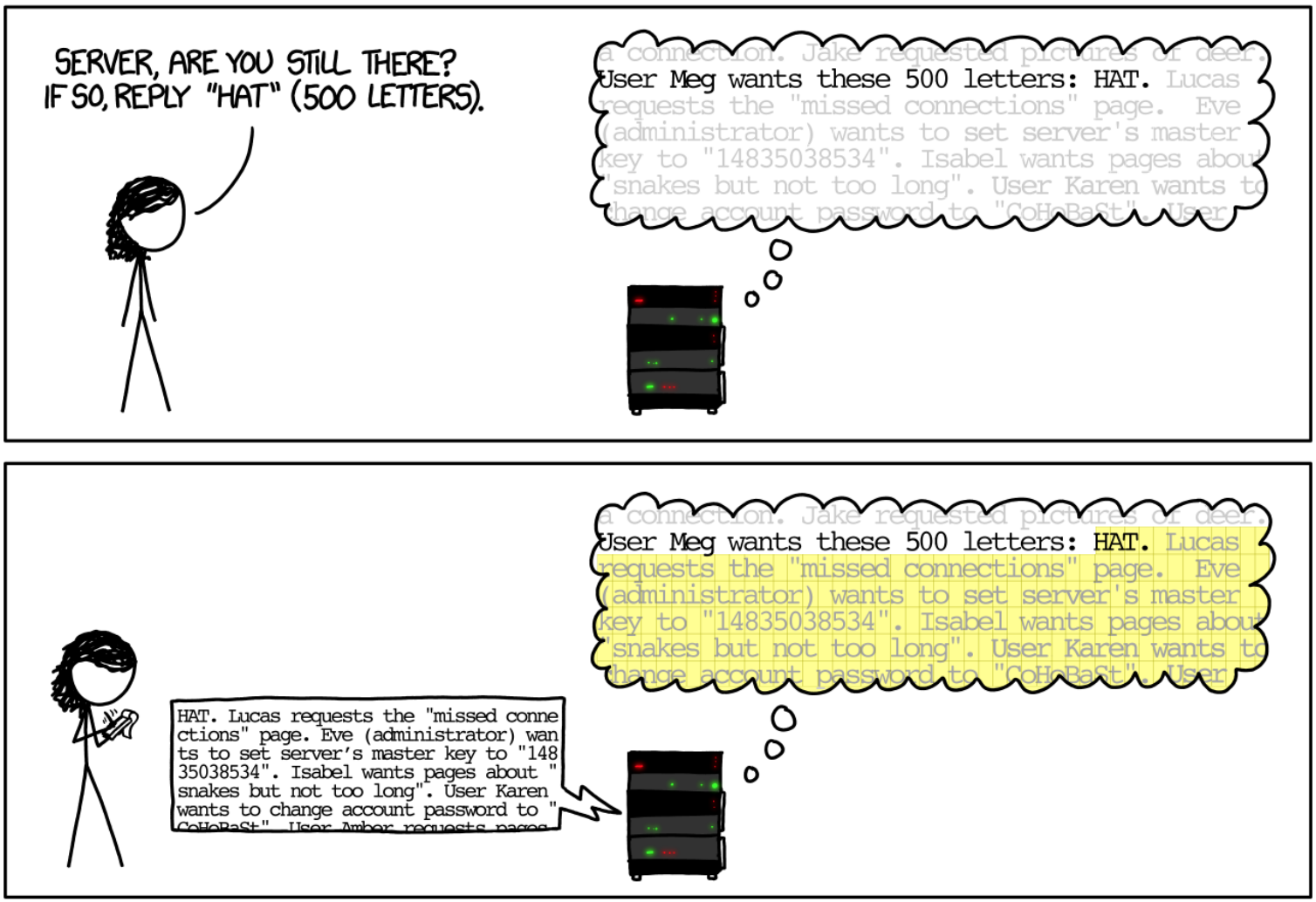
Softwarekatastrophe Heartbleed



Softwarekatastrophe Heartbleed

- 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



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"

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

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

Djikstra (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.

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

Probleme bei der Softwareentwicklung

- 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.

Ziele des Software Engineerings

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)

Phasen des Software Engineering

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

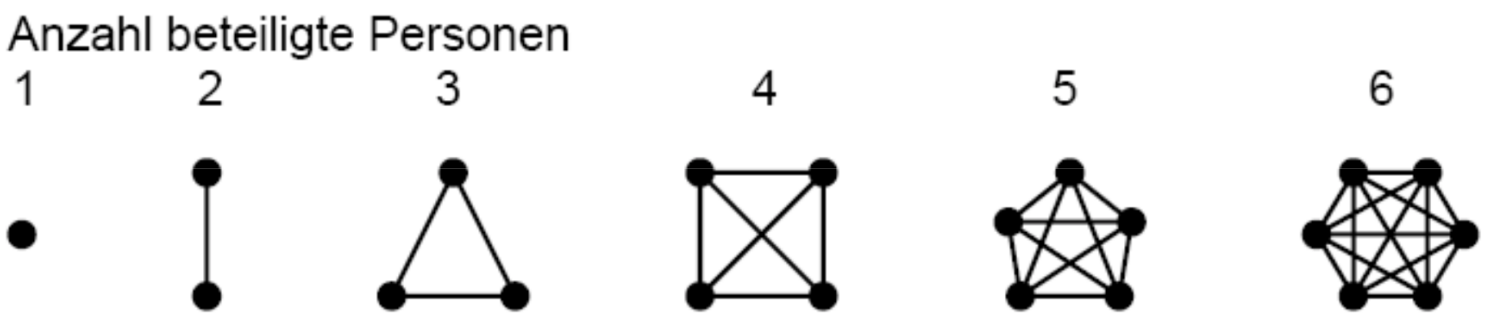
Kleine Softwareprojekte

- 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

Große Softwareprojekte

- 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 erfoderlich
- Projektstruktur zwingend erforderlich

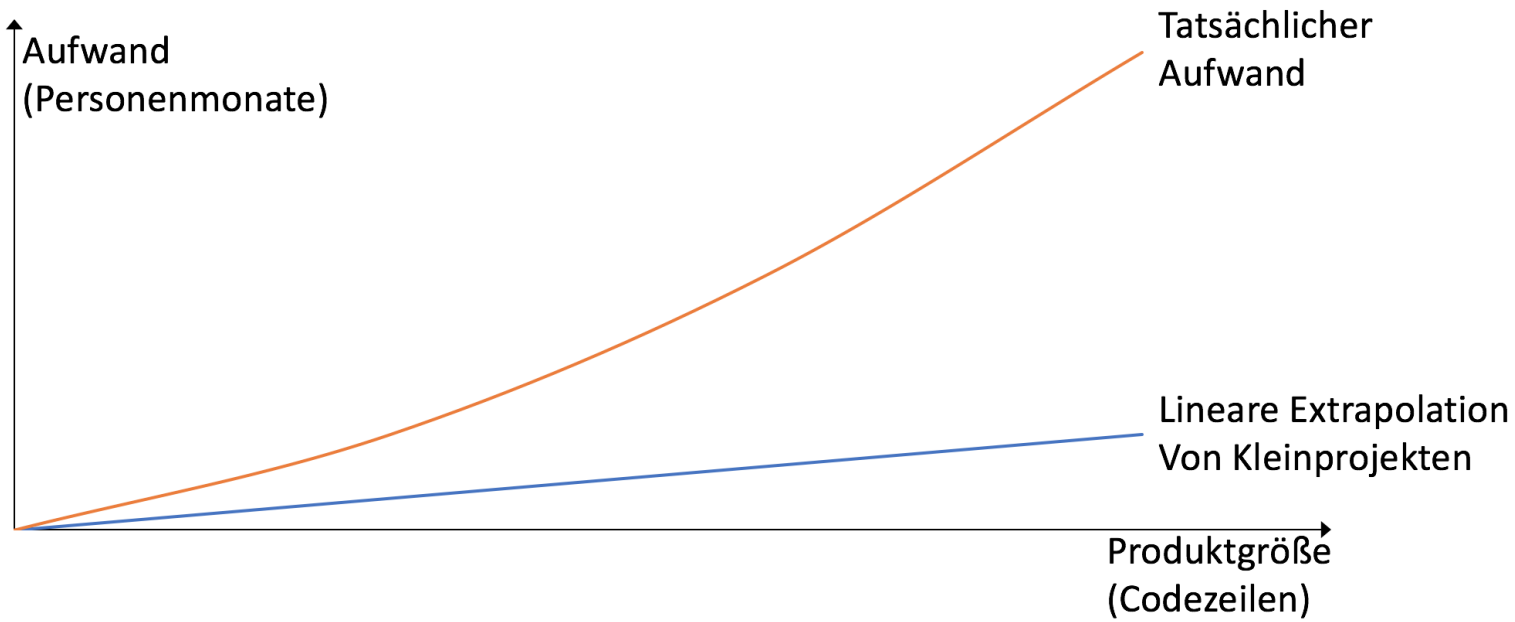
Wachstum des Kommunikationsbedarfs



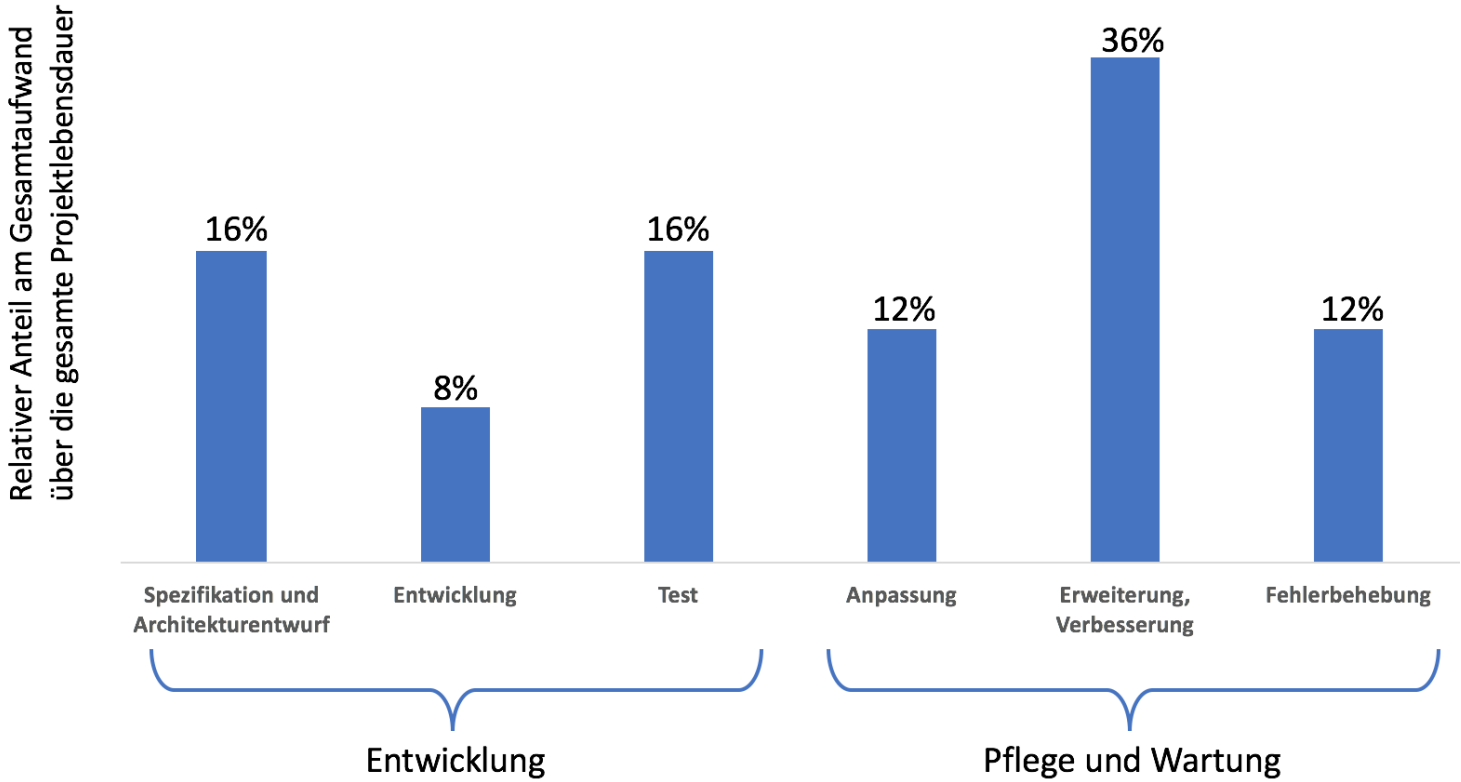
Quantensprung: Kommunikation wird erforderlich

Quantensprung: Zahl der Kommunikationspfade übersteigt Zahl der Personen

Wachstum des Aufwands



Aufwände aufgeschlüsselt



Übung 1.1

Eine Person braucht zum Bau einer 2m langen Bruecke 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?

Übung 1.1 - Lösung

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

1. Aufwand = (2000m / 2m * 0.5PT) / 100 Personen = 5 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.

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.

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.

Softwareentwicklungsprozesse in der Praxis

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

Übung 2.1

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-2-Market
5. Art der Software die entwickelt wird
6. Kundentyp
7. ...

Übung 2.1 - Lösung

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. ...

Prozessmodell vs konkreter Prozess

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

Beispiele Modell vs konkreter Gegenstand

Modell	Gegenstand
Theaterstück	Aufführung
Musik-CD	Einmalige Wiedergabe
Applikation	Ausführung der Applikation
Klasse	Objekt
Vorgehensmodell	Projektablauf
Prozessmodell	Projekt (inkl. Organisation)

Merkmale von Modellen

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

Modelle beim Software Engineering

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

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.

Code and Fix

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

Code and Fix

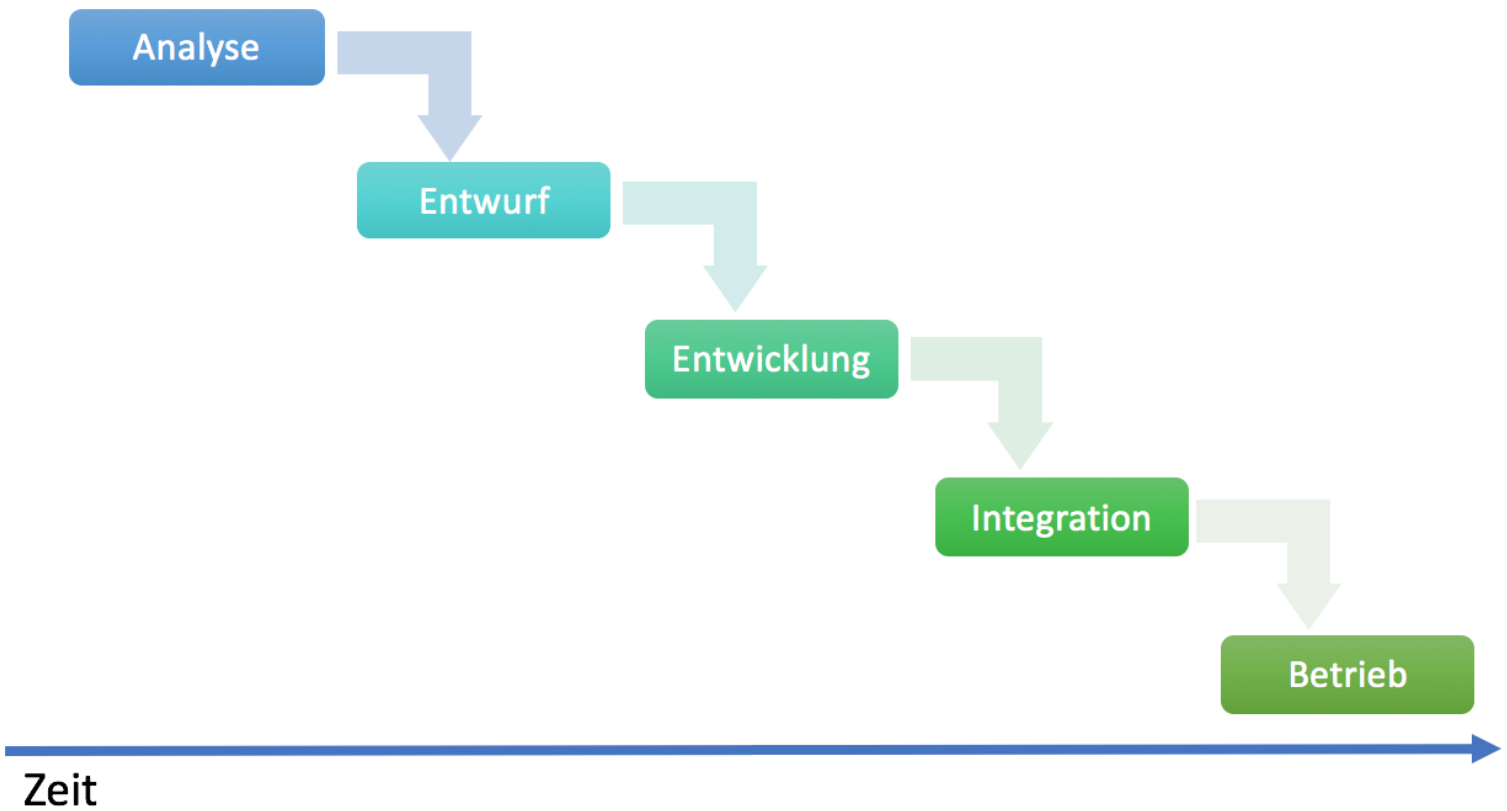
Nachteile

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

Um die Nachteile zu umgehen werden zusätzliche Aktivitäten benötigt

Sequenzielles Modell

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



Sequenzielles Modell

Vorteile

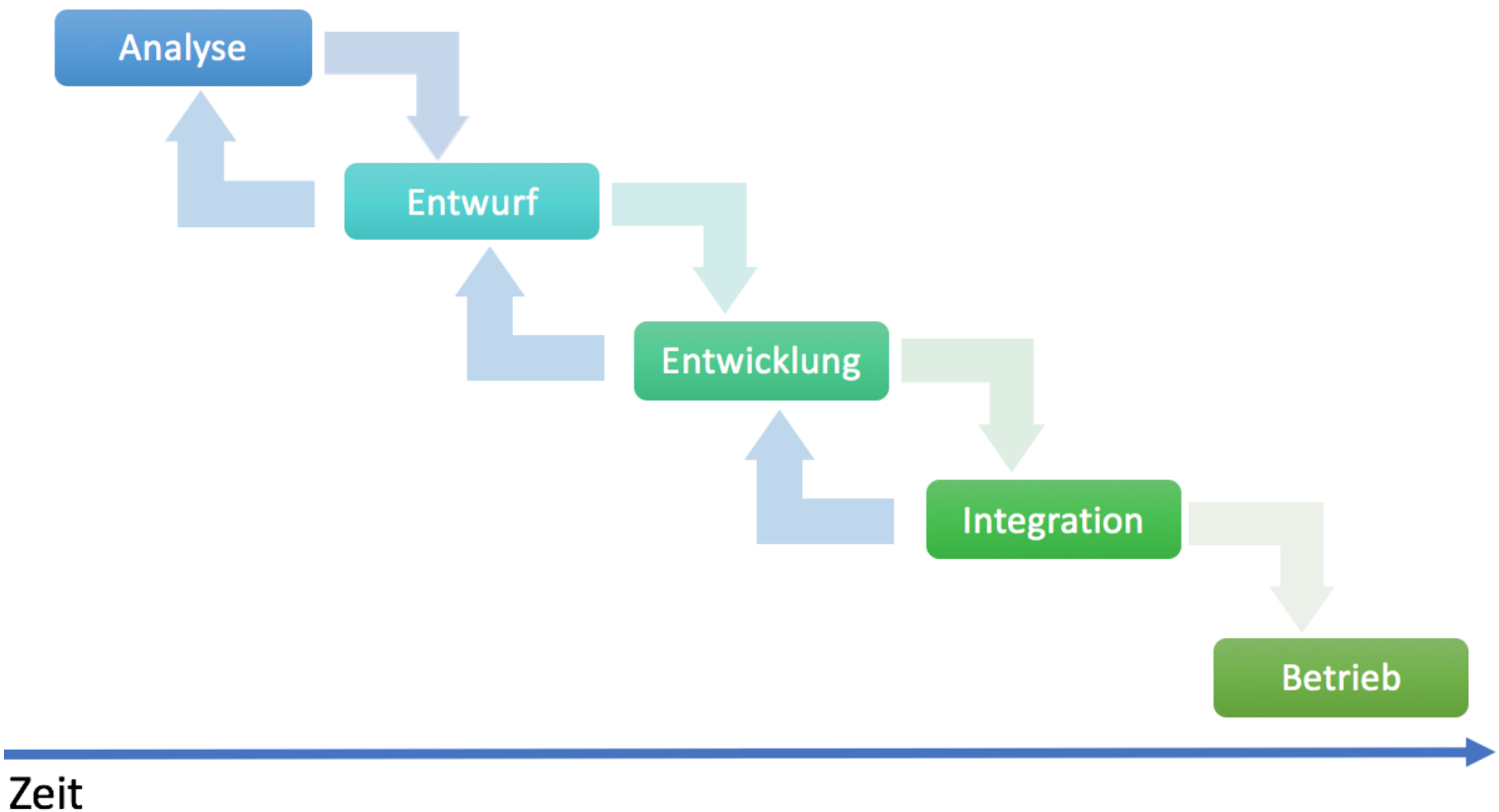
- leicht verständlich
- geringer Managementaufwand

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

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



Wasserfallmodell als bekanntestes sequentielles Modell

Vorteile

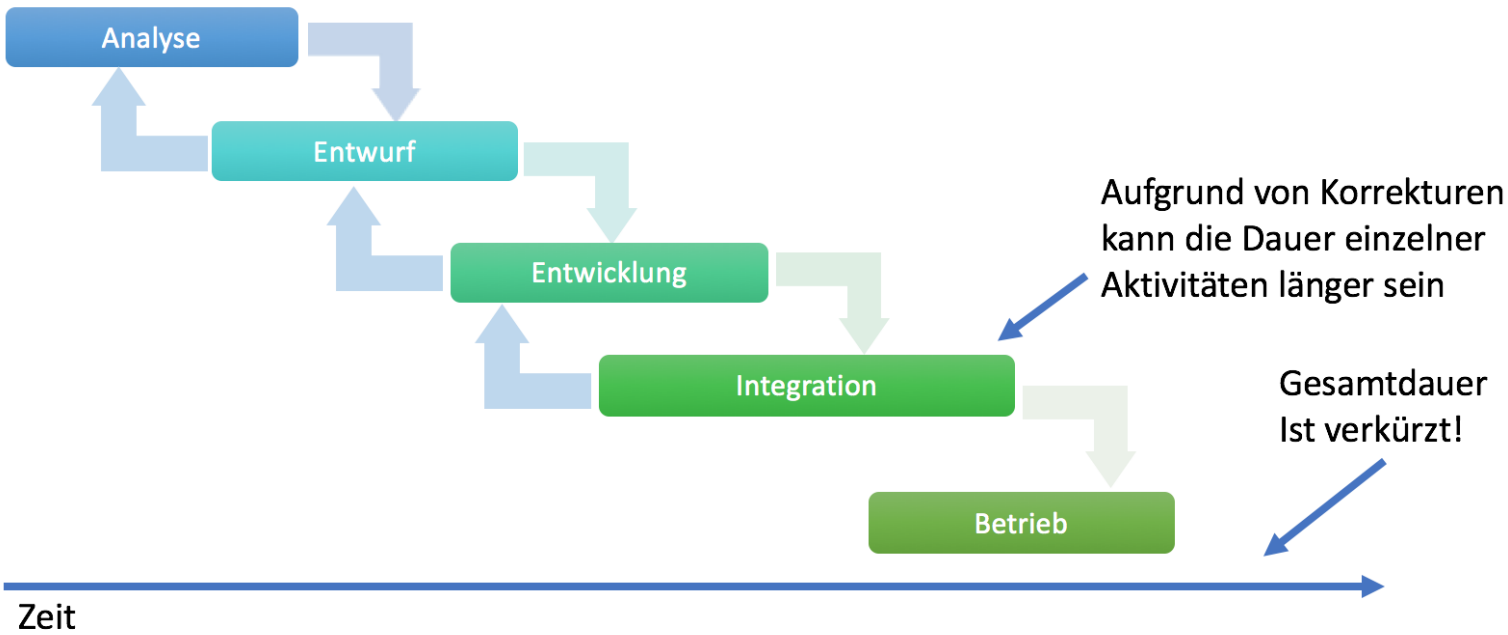
- leicht verständlich
- geringer Managementaufwand
- Aktivitäten gut dokumentiert

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

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



Nebenläufiges Modell

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

Inkrementelles Modell

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

Inkrementelles Modell

- 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

Inkrementelles Modell

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

Evolutionäres Modell

- 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

Evolutionäres Modell

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 Änderugen 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

Übung 2.2

Wann würden Sie evolutionäre Modelle anwenden und wann sequenzielle?

Anwendung evolutionärer Modelle

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

Übung 2.3

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 Vorgängerpahse 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?

Übung 2.3 - Lösung

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

Übung 2.4

Sie arbeiten nun nach dem inkrementellen Vorgehensmodell. Die Analyse benötigt 2 Monate. Sie entwicklen 2 inkremente (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?

Übung 2.4 - Lösung

V1 = 2M A + 1M E + 1M C = 4 Monate
V2 = 1M E + 1M C = 2 Monate
V3 = V1 + V2 = 6 Monate

Vorteile

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

Übung 2.5

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?

Übung 2.5 - Lösung

V1 = 1M A + 1M E + 1M C = 3 Monate
V2 = 1M A + 1M E + 1M C = 3 Monate
V3 = V1 + V2 = 6 Monate

Vorteile

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

Prototyping Modell

- 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.

Prototyping Modell - Arten von Prototypen

Prototypen haben die folgene 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

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

Prototyping Modell - Arten von Prototypen

Labormuster

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

Pilotsystem

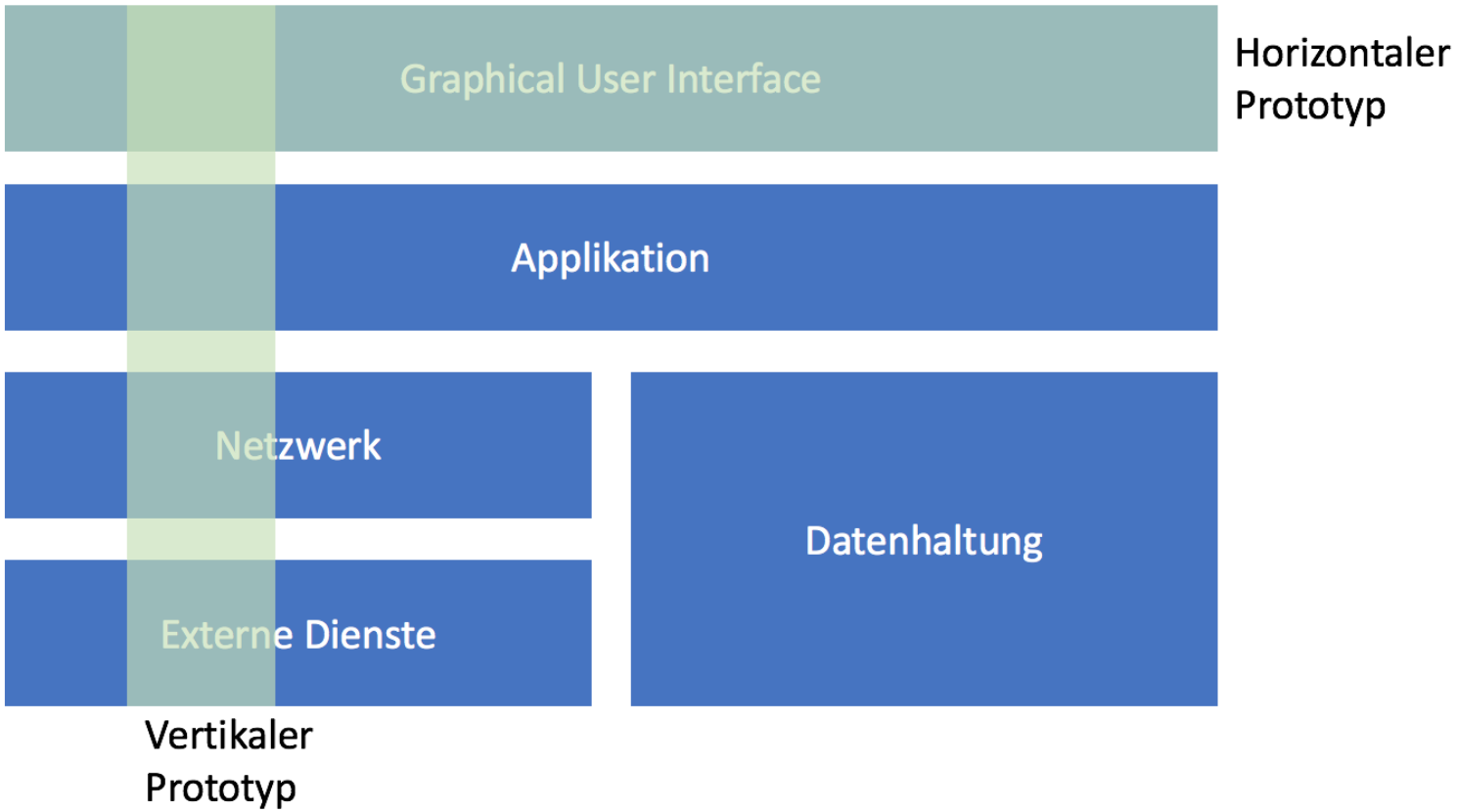
- Realisiert einen abgeschlossenen Bereich des Zielsystems
- Funktionalität und Qualität reichen mindestens für vorübergehenden Produktiveinsatz
- Wird Schrittweise erweitert

Prototyping Modell - Arten von Prototypen

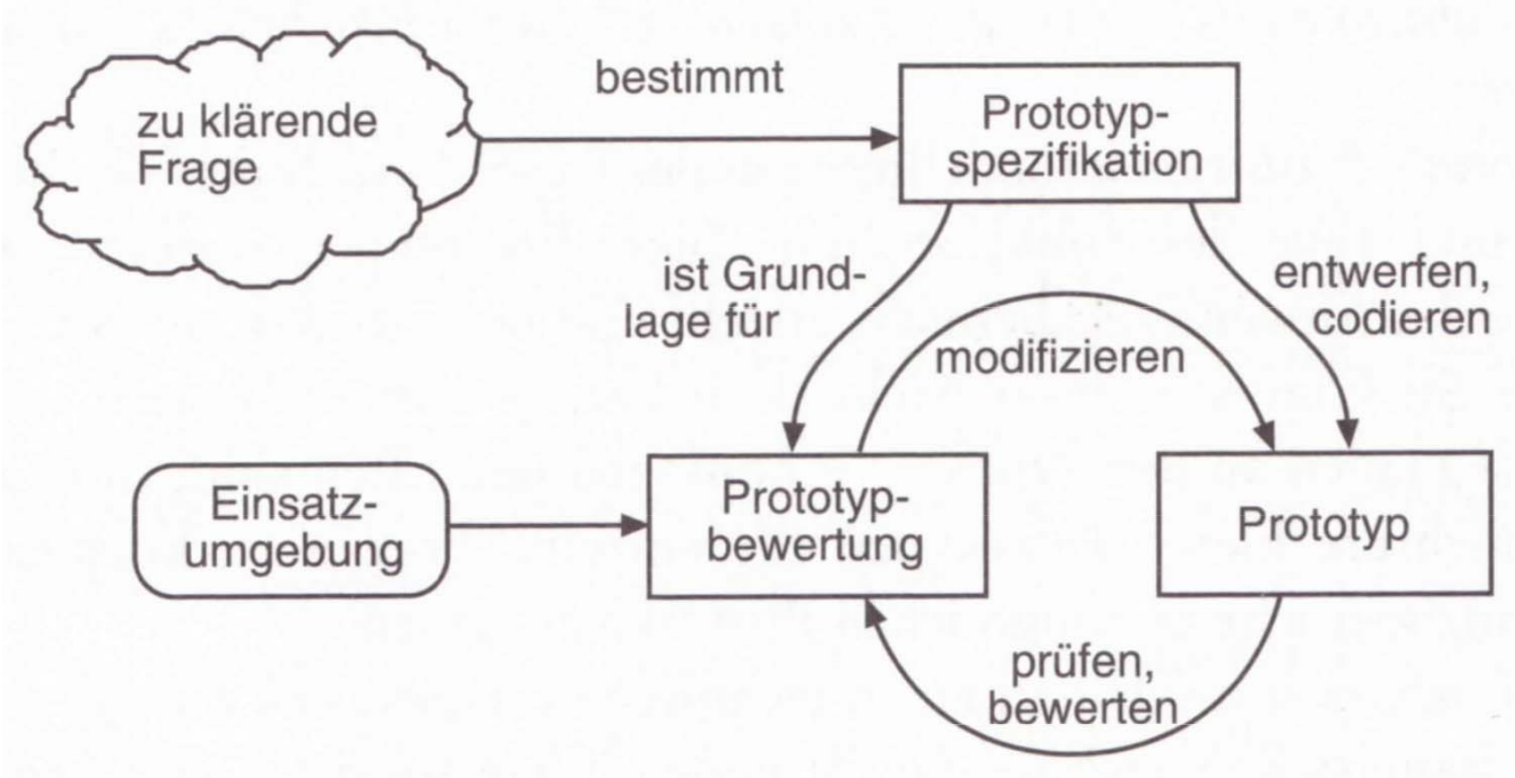
Ein Softwaresystem besteht in der Regel aus verschiedenen Komponenten/Schichten. Aus diesem Grund lassen sich Prototypen auch in horizontale oder vertikale Prototypen unterscheiden.

- 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)

Prototyping Modell - Arten von Prototypen



Prototyping Modell - Prozess



Prototyping Modell - Anwendbarkeit

- 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.

Übung 2.6

Welche Vor- und Nachteile hat die Verwendung von Prototypen?

Prototyping Modell

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

Spiralmodell

Monumentale Prozessmodelle

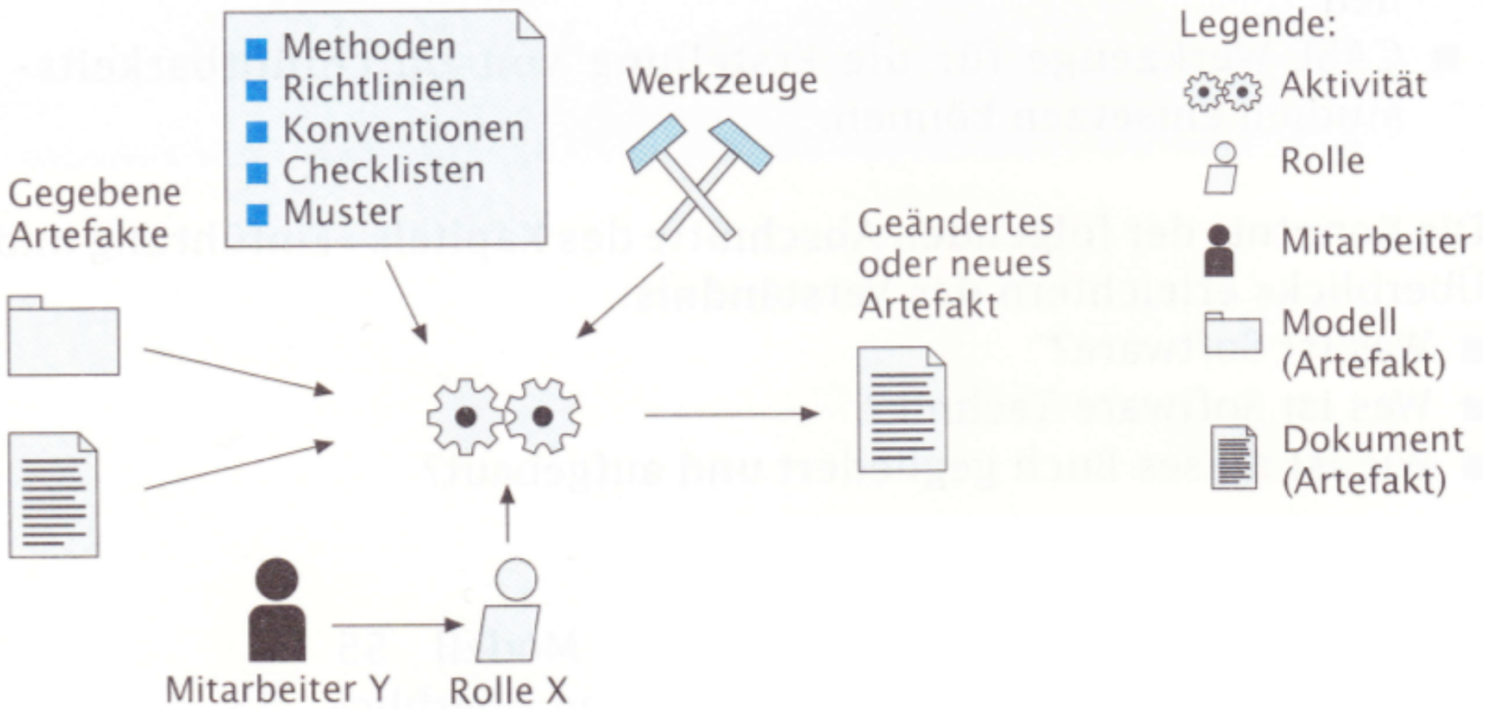
Definition Prozessmodell

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

Monumentale Prozessmodelle

- Aktivitäten werden von Mitarbeitern durchgeführt
- Die Kenntnisse/Fähgikeiten die als Vorraussetzung 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

Monumentale Prozessmodelle



V-Modell

Rational Unified Process

Agile Prozessmodelle

Agile Prozessmodelle

Extreme Programming

Scrum

Kanban

Übung 2.X

Wann sollten sequenzielle und wann inkrementelle Modelle eingesetzt werden?

Projektmanagement

Projektplanung

Qualitaetssicherung

Messen und Bewerten

Requirements Engineering

Software Architektur

Konfigurationsmanagement