

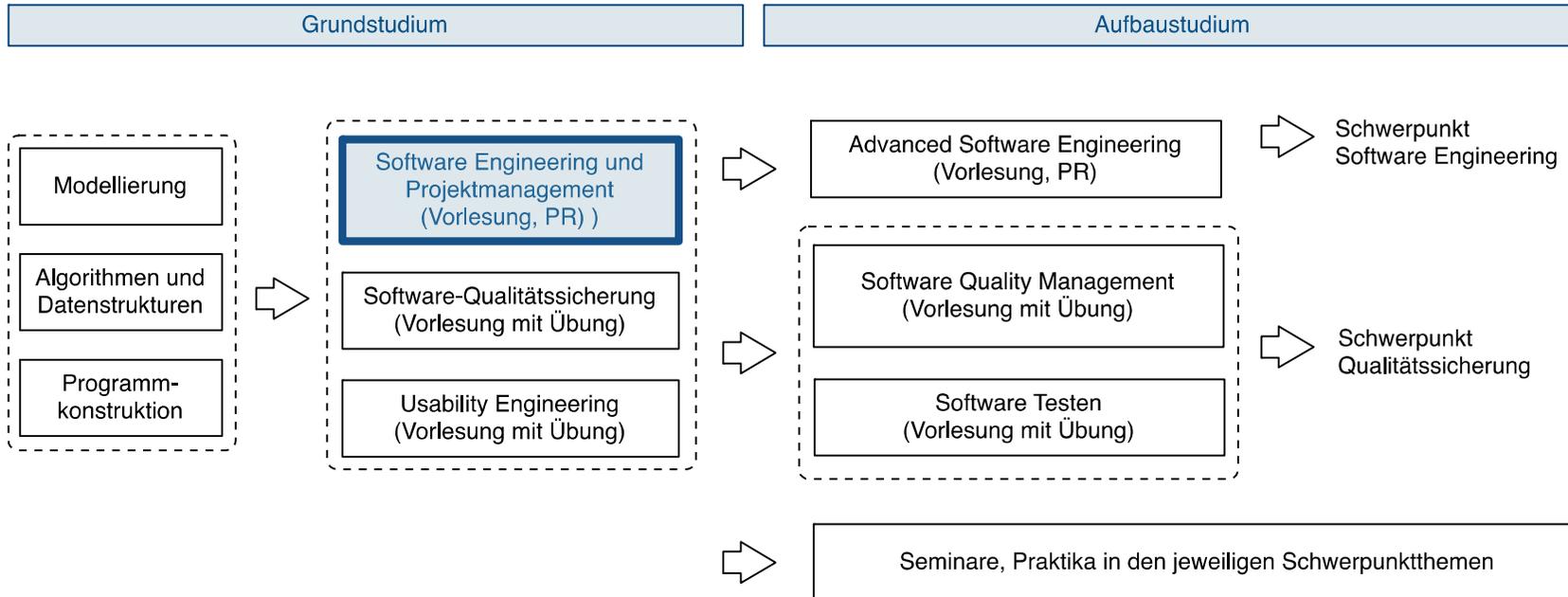
Software Engineering & Projektmanagement (SEPM) Vorlesung Block „Einführung in SE“

Stefan Biffi

Grundlagen SE-Projekte
Projekttypen
Software Prozesse und Produkte



Software Engineering im Studium



Featured Project *Presentr*

Web-Plattform für effiziente interaktive Präsentationen

- *Presentr* ist ein aus einem erfolgreichen Projekt aus *Advanced Software Engineering* entstanden.
- Beiträge von *Presentr* (<http://presentr.at>)
 - Vortragende können via *Presentr* im Browser präsentieren.
 - Studierende können an Audience Response Tasks teilnehmen.
 - Studierende können die Präsentation mit den Ergebnissen des Audience Response Tasks als PDF herunterladen.
- Dadurch können Vorträge effizienter interaktiv gestaltet werden.
- *Presentr* wird anhand des Feedbacks kontinuierlich weiter entwickelt.
- *Presentr* wird in diesem Semester in der SEPM Vorlesung durchgehend verwendet.
- Ihre Rückmeldungen sollen in die Gestaltung von *Presentr* einfließen.

Ziele der Vorlesung



- **Ergänzung zum SEPM Projekt**
 - Ein Team, ein mittelgroßes Produkt, ein Prozess.
 - -> Unterschiedliche Projekttypen und -größen.

- **Erlernen wesentlicher Konzepte der Software-Entwicklung**
 - Technische Grundlagen.
 - Ergänzungen/Erweiterungen zu Inhalten aus dem Projekt.
 - Aktuelle Methoden in der Software-Entwicklung.
 - Software Life Cycle.
 - Vorgehensmodelle.
 - Projektmanagement.

Inhalte 1/2

1. Einführung in Software Engineering

- Projekttypen: Eingebettete Systeme, kommerzielle Software
- Umfeld von Software Engineering (Personen, Projekt, Markt, Technologie)

2. Projektmanagement Teil 1

- Projektauftrag, Umfeldanalyse
- Strukturpläne und Planungsablauf

3. Techniken und Werkzeuge

- Technische Grundlagen (Configuration Management, Source Code Management, Build Management)
- Standardtechnologien für die Projektorganisation

4. Modellierung von Anwendungsszenarien

- Grundlagen (Modell & Diagramme, Bedarf an Modellierung)
- Daten und Kontrollflussmodelle in UML

5. SE Phasen

- Anforderungen, Entwurf & Design, Implementierung, QS & Testen, Wartung



Inhalte 2/2

6. SE Prozesse im Überblick

- Software Life-Cycle, Vorgehensmodelle (Strukturierte & Agile Ansätze)

7. Software Design

- Architekturevaluierung, Tests
- Design Patterns

8. Projektmanagement Teil 2

- Produktionsfaktor Mensch
- Auswahl von Mitarbeitern, Team Management

9. Persistenz-Strategien

- Objektorientierte und Relationale Datenbanken
- Web-Services/REST, Cloud Computing

10. Qualitätssicherung in SE Projekten

- Review von SE-Modellen (ER, UML)
- Erstellen, Beurteilen und Anwenden von Testfällen
- Test Driven Development (TDD)

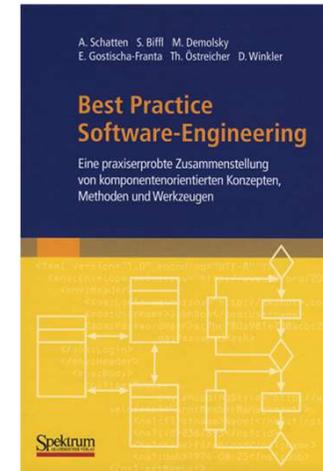
Vorlesungsprüfung



Die Prüfung besteht aus

- 10 Theoriefragen (60 Punkte)
- 1 Kreativbeispiel (40 Punkte)
Das Kreativbeispiel kann kleine Modellierungsbeispiele und andere praktische Aufgaben umfassen.
- Die Benotung orientiert sich am üblichen Schema.
- Beide Teile der Prüfung (Theorie- und Kreativteil) müssen positiv sein.

- Termin: Juni (Nebentermin April)
(Anmeldung via TISS ab ca. 1 Monat vor der Prüfung)
- Prüfungsgrundlage sind die in der Vorlesung besprochenen Inhalte (Folien, BPSE Buch).
- Closed Book Prüfung!
- Empfohlene Literatur siehe TUWEL.
- Beispiele bisheriger Prüfungen: siehe Prüfungsordner im TUWEL.



Überblick Block 1: Einführung, SE-Projekte



Block 1-1 “Einführung” (30’)

- BPSE Buch, Kapitel 1
- Grundlagen, Begriffe
- Begleitendes Beispiel zur Vorlesung: e-Katalog

Block 1-2 “SE-Projekte” (45’)

- Projekttypen, Umfeld des SE-Prozesses
- Größenordnungen von Projekten
- Faktoren für ein erfolgreiches Projekt

Block 1-3 “Produkte und Prozesse” (45’)

- Produkte in der Software-Entwicklung
- Vorgehensmodelle



Motivation - Ziel



- Software ist zunehmend Teil des täglichen Lebens.
- Die Herstellung **komplexer Software** ist anspruchsvoll und braucht professionelles Herangehen: Software Engineering.
- Fehler und Qualitätsmängel betreffen immer mehr Menschen in immer weitergehendem Umfang: Ziel „**No surprise**“- Software.
- Herausforderungen: Heterogenität, Änderungen von Anforderungen, Sicherheit und Vertrauen, Skalierung
- Software Engineering soll helfen, für große Software Systeme ähnliche Qualitätsmaßstäbe zu erreichen wie in klassischen Ingenieursdisziplinen.
 - Kostengünstige Entwicklung
 - Hohe Qualität
 - Innerhalb des geplanten Zeitrahmens



Produkt- und Projekttypen



- Unterschiedliche Projekte verlangen unterschiedliche **Vorgehensweisen**.

Typ	Anforderung	Beispiel
Kommerzielle Anwendungen (interaktiv, Transaktionen)	Benutzbarkeit, Verfügbarkeit, Support	Datenbanktransaktionen, Finanz-/Touristikanwendg.
Eingebettete Systeme (Steuerungssysteme)	Zeitgesteuert, Sicherheit, Echtzeitanforderungen	Handy, ABS-System, Liftsteuerung, Kraftwerk, Fabriksteuerung
Wissenschaftliche Software (Systeme zur Datenanalyse)	Rechengenauigkeit, Korrektheit, Zuverlässigkeit	Medizinische oder Luftfahrtprogramme Analyse großer Datenmengen, Vorhersagen

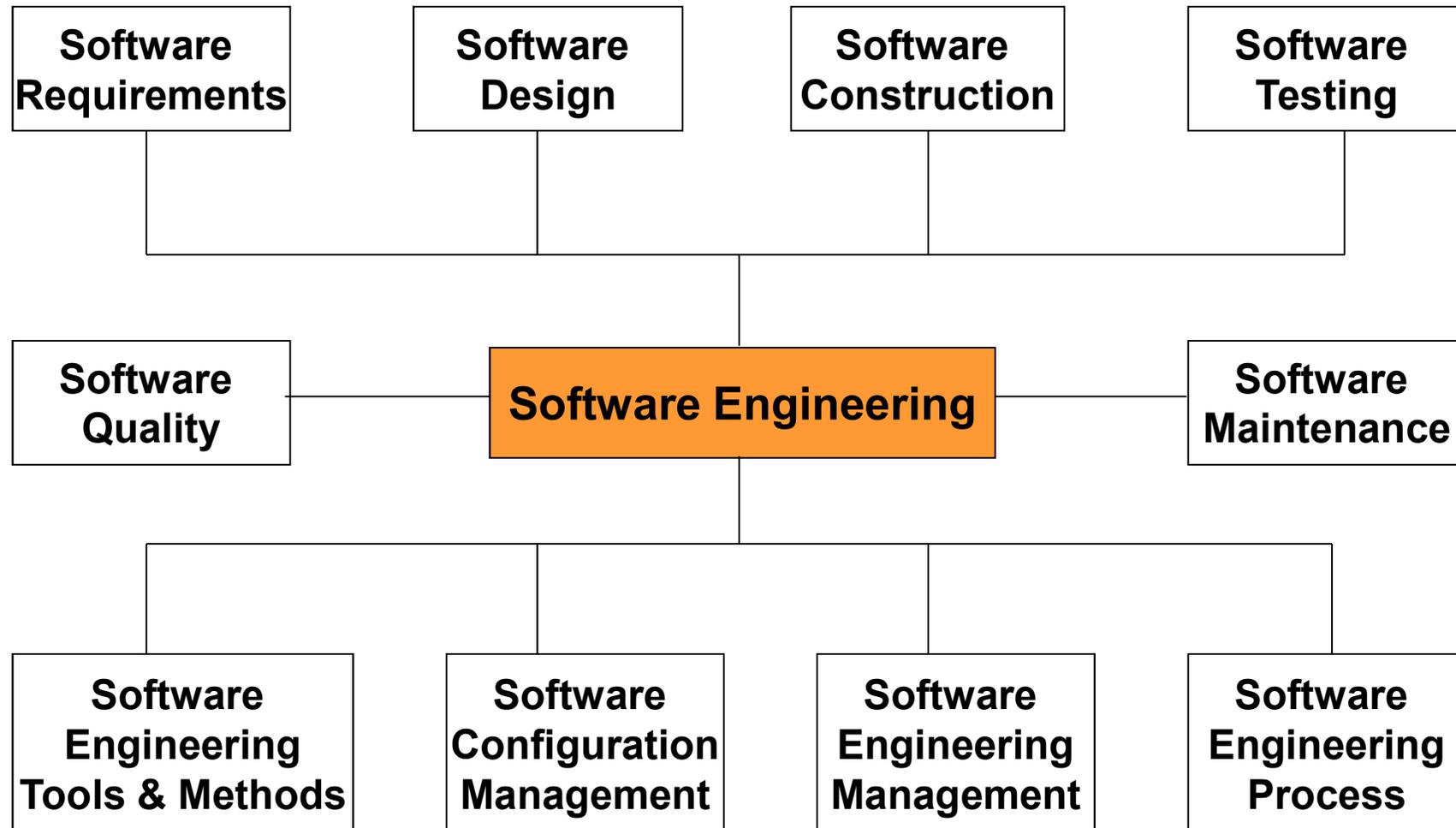
Begriffsdefinitionen



- **Software:** Beinhaltet alle Instruktionen die dem Computer vorschreiben, was er zu tun hat. **Programme**, die die Hardware des Computers kontrollieren.
- **Software Engineering:** **Vorgehensweise** zur **systematischen Erstellung** von Software nach „ingenieurmäßigen Prinzipien“.
 - Systematisch, effektiv,effizient.
- **Analyse:** Erfassen und Verarbeiten von **Anforderungen** (Pflichtenheft).
- **Design:** Entwurf der internen **Struktur des Systems** (Systemarchitektur).
- **Validierung:** Überprüfung des Verhaltens eines Programms durch festgesetzte Testfälle.

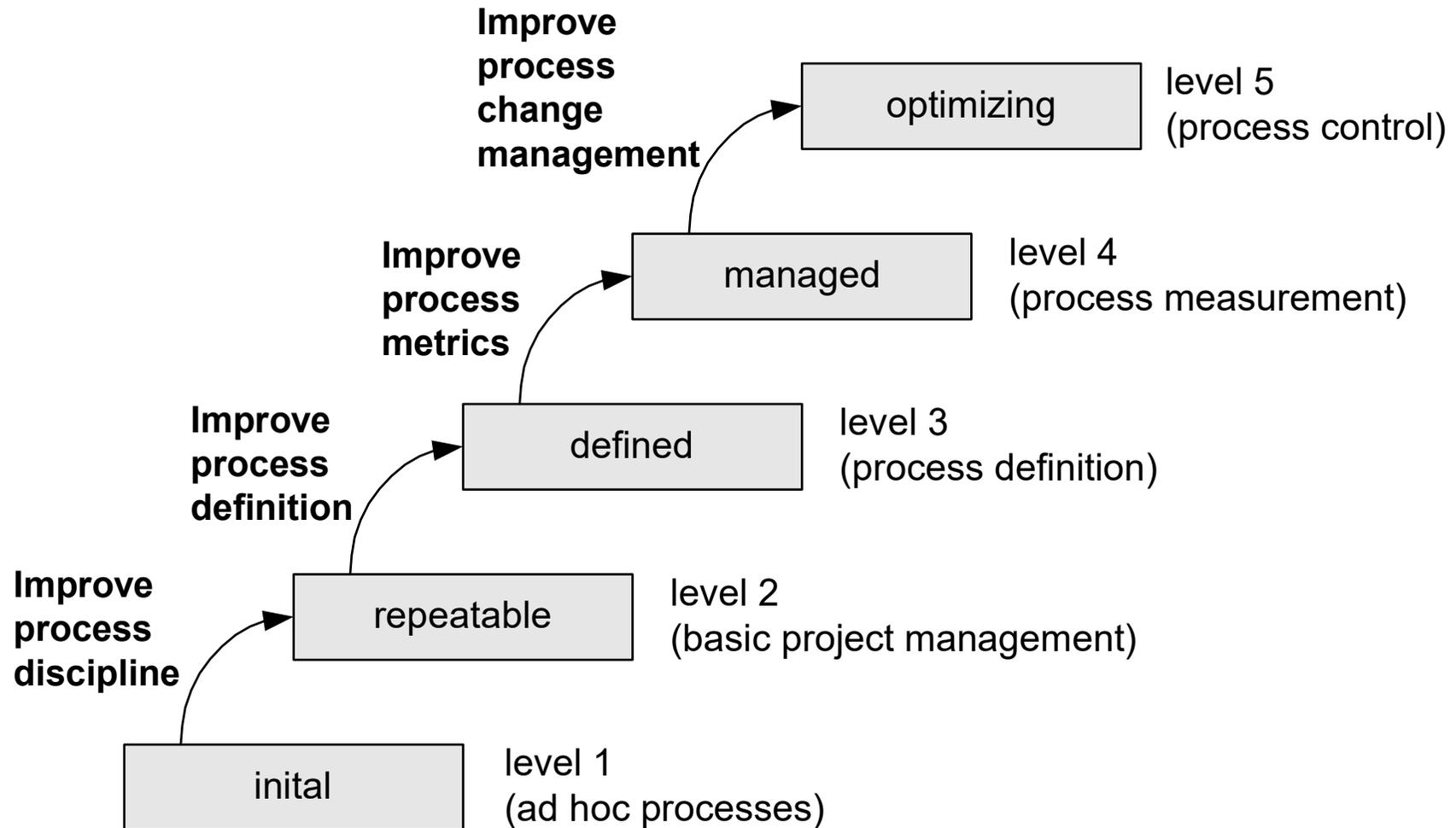
Software Engineering – Teilbereiche

SWEBOK.org



Reifgrade der Software-Entwicklung

Capability Maturity Model Integrated CMMI



CMMI - <http://www.sei.cmu.edu/cmmi/>

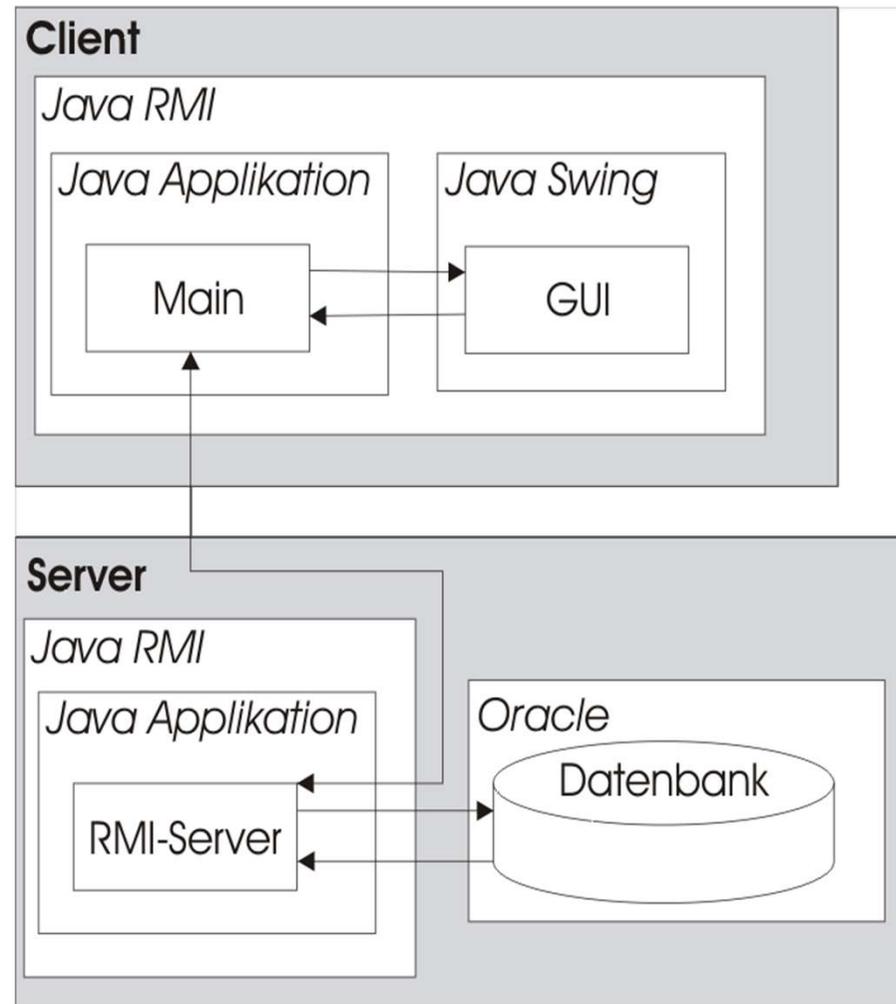
CMMI – Levels 1, 2, and 3



- **Level 1 (initial)**
 - ad-hoc processes („like it is“)
- **Level 2 (repeatable – „basic project management“)**
 - Configuration Management
 - **Software Quality Assurance**
 - Subcontract Management
 - *Project Tracking and Oversight*
 - *Project Planning*
 - **Requirements Management**
- **Level 3 (defined – process definition)**
 - **Peer Reviews**
 - Intergroup Coordination
 - Software Product Engineering
 - Integrated Software Management
 - Training Programs
 - Organization Process Definition
 - Organization Process Focus

Architekturbeispiel – Kleines Programm

- Wenige Komponenten
- Wenige Technologien
- Einfache Architektur

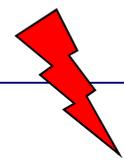


Beispiel: Mangelhafte Software

22. Juli 1962, Cape Canaveral / Florida:

Fehlstart der ersten amerikanischen Venussonde Mariner 1.

Ausschnitt aus dem FORTRAN – Programm zur Steuerung der Trägerrakete (NASA):



```
IF (TVAL .LT. 0.2E-2) GOTO 40
DO 40 M = 1, 3
    W0 = (M-1)*0.5
    ...
    DO 5 K = 1.3
        T(K) = W0
        Z = 1.0/(X**2)*B1**2+3.0977E-4*B0**2
        D(K) = 3.076E-2*2.0*(1.0/X*B0*B1+3.0977E-4(B0**2-
X*B0*B1))/Z
        E(K) = H**2*93.2943*W0/SIN(W0)*Z
        H = D(K)-E(K)
    5 CONTINUE
    10 CONTINUE
    Y = H/W0-1
40 CONTINUE
```


Typische Probleme großer SW-Projekte



- Zeitverzögerung bei Software-Projekten
- Überschreiten des geplanten Budgets
- Mangelnde Qualität der Software
- Spätes Erkennen von Design-Fehlern („late design breakage“)
- Schwierige und teure Wartung

SE/QM-Ansätze

- Formale Methoden: Verifikation, bessere Programmiersprachen
- Prozessverbesserung: Produkte, Prozesse, Vorgehensmodelle
- Personen: Training, Motivation, „Kultur guter Arbeit“

Software Engineering Vorlesung

Einheit 1-2 Projekttypen



- Überblick Projekttypen
- Größenordnung von Projekten
- Umfeld des SE-Prozesses
- Software Beschaffung
- Faktoren für ein erfolgreiches Projekt



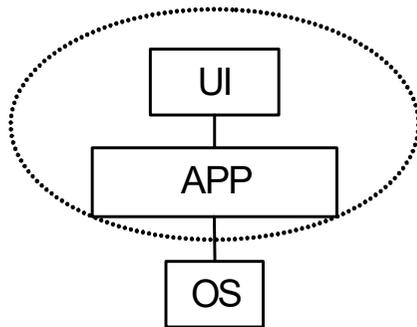
Motivation - Ziel



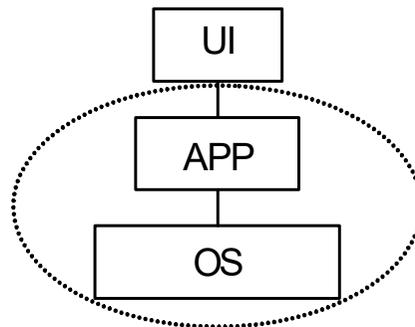
- Software-Herstellung ist nicht-trivial und braucht professionelles Herangehen
- Unterschiedliche Projekte verlangen unterschiedliche Vorgehensweisen
- Komplexität - Einfluss der Projektgröße
- Design-Strategien: Software kaufen oder selber entwickeln
- Risikofaktoren - Warum scheitern viele Projekte
- Erfolgsfaktoren - Was sind Grundlagen für ein erfolgreiches Projekt

Einige Projekttypen

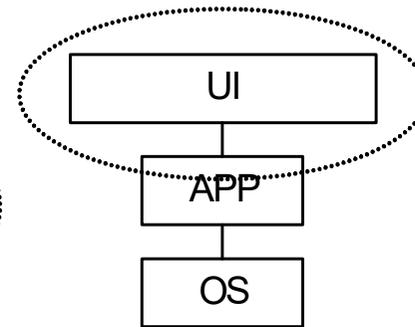
- Ein Software-System besteht typischerweise aus (a) **Benutzerschnittstelle** (UI), (b) **Geschäftslogik** der Anwendung (App) und (c) einem **Betriebssystem** (OS).
- Je nach Software-Typ ist der Schwerpunkt der Entwicklung unterschiedlich.



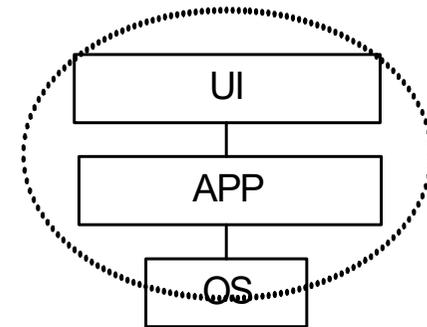
a) commercial system



b) real-time system



c) Web application



d) computer games

UI ... User Interface (Benutzerschnittstelle)
APP ... Application (Anwendung)
OS ... Operating System (Betriebssystem)

Komplexitätstreiber (PM & SE)



Merkmal	Messbare Attribute	Beispiel
Größe (PM)	Anzahl der beteiligten Personen	Personen: 50
Dauer und Aufwand (PM)	Anzahl Wochen bzw. Monate	Dauer: 52 Wochen Personen-Jahre: 3 PJ
Verwendete Technologien (SE)	Art, Anzahl und Alter der Technologie	Art: Scriping language, Alter: 4 J., Anzahl: 5
Komplexität (SE)	Anzahl Klassen, Module, Datenbanken; verwendete Technologien, Zeilen Code	Datenbanken: 2 Klassen: 42 Code: 30 000 Zeilen

.....

Projekttypen – Größe eines Projekts



Größe	Kriterien	Beispiele
Klein	Bis zu 6 Personen Monate: 0-8 Anzahl Technologien: <5	Rechenprobleme, Algorithmen
Mittel	10-30 Personen Monate: 9-24 Anzahl Technologien: 5-12	Buchhaltung, Lagerverwaltung
Groß	50-100 Personen Monate: 25-45 Anzahl Technologien: 12-20	Compiler, Datenbank
Sehr groß	Ab 100 Personen Monate: 45-n Anzahl Technologien: >20	Raumfahrt, Atomkraftwerk, elektronische Börse, große Standardsoftware

.....

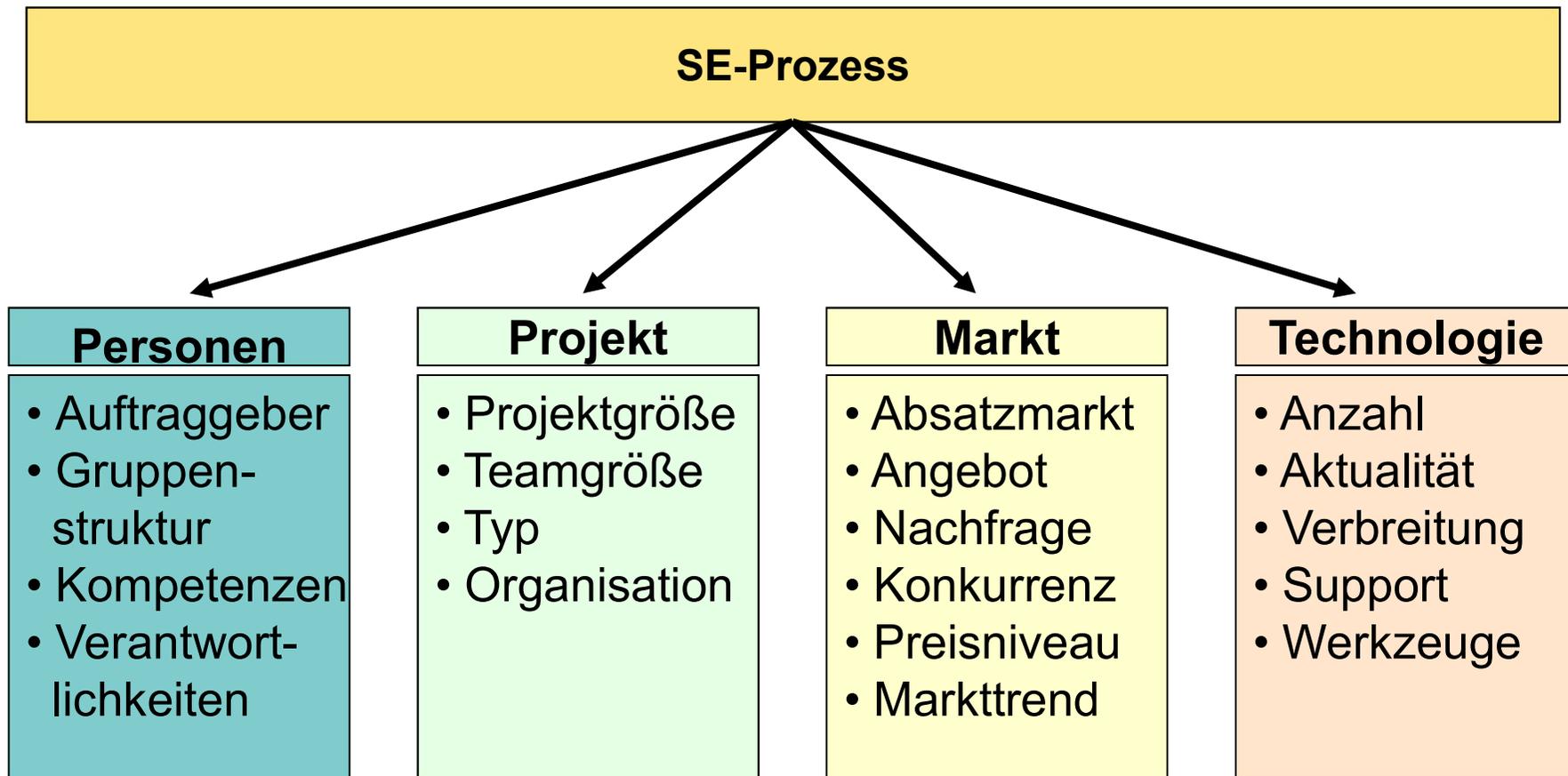
Was ändert sich mit Zunahme der Größe



- Komplexität (SE, PM)
- Bedarf an Flexibilität (SE, PM)
- Bedarf an Organisation, Planung und Überblick steigt (PM)
- Bedarf an Prozessorientierung (PM)
- Human Factors – Bedarf an Kompetenzen steigt (PM)

- Kommunikationspfade – wer kommuniziert mit wem (SE, PM)
- Testaufwand steigt (SE)
- Saubere Dokumentation ungleich wichtiger, (SE)
damit das Projekt nachvollziehbar bleibt, vor allem die Schnittstellen
- Versionenverwaltung bzw. Konfigurationsmanagement
damit kein Versions-Chaos entsteht (SE)

Umfeld des Software Engineering Prozesses



Software Beschaffung 1/2

Kauf oder Entwicklung



	Kauf	Abänderung	Neuerstellung
Entspricht Anforderungen	Ungefähr (oft viele ungenützte Funktionen oder ein paar fehlende)	Oft ist keine exakte Anpassung möglich → Beschränkung durch technische Grenzen	Genau
Änderbarkeit	Schwierig, da technische Details oft nicht transparent sind	Ausgangsprodukt wurde selbst hergestellt: leicht änderbar, Durch andere hergestellt: Schwer änderbar	Gut möglich (Dokumentation der Entwicklung und somit techn. Transparenz vorhanden)
Preis	Nach Anforderung + Verbreitung	- durch eigene IT- Abteilung: kalkulierbar - durch andere: kostenintensiver	Teuer

Software Beschaffung 2/2

Kauf oder Entwicklung



Kauf von Software, wenn ...

- die **gewünschten Anforderungen** weitgehend erfüllt werden.
- die **individuelle Anpassung** (leicht) möglich ist.
- die **Kosten** niedriger sind als bei Eigenentwicklung.
- ausführliche **Dokumentation** vorhanden ist.
- guter **Support** bei Problemen und Fragenstellungen geboten wird.

Vergleich zwischen kommerzieller Software und eingebetteten Systemen



	Embedded Systems	Kommerzielle Software
Steuerung	Ereignisgesteuert, oft auch vollständig automatisiert	Benutzergesteuert
Kosten	Teuer, wegen Neuentwicklung oder Anpassung	Kaufen billiger als selber entwickeln
Zuverlässigkeit	Sehr wichtig	Oft nicht entscheidend
Wartung	Schwierig, z.T. Hardware-technisch unmöglich	Meist professioneller Support
Sicherheit	Müssen sicher sein (<i>safety</i>)	Unterschiedliche Wichtigkeit: Online-Banking, Datenbank Systeme vs. Photobearbeitung
Usability	Benutzerinterface rudimentär oder nicht vorhanden	Oft entscheidend, vor allem bei großer Konkurrenz
Beispiele	Handysteuering, Liftsteuerung, ABS-System, Ampel	Datenbanksystem, Web-Applikationen, Texteditor

Beispiel Eingebettetes System



Liftsteuerung, Fabriksteuerung

➤ Ereignisgesteuert

- Knopfdruck
- Sensoren-Ereignis, wenn Stockwerk erreicht wird

➤ Sicherheit

- System muss absolut stabil sein
- Verschiedene Sicherheitsmaßnahmen bei Notfällen

➤ Kostenfreundlich

- Berechnung des kürzesten Weges bei mehreren Anfragen
- Standby Modus, wenn keine Anfragen

➤ Usability

- Einfache, leicht verständliche Benutzersteuerung
- Logische Prozessabfolge



E- Katalog als VO begleitendes Beispiel 1/2



- Zu realisieren ist ein online Katalog, in dem die Produkte ...
 - benutzerfreundlich dargestellt werden.
 - selektiert werden können.
 - in einem "Warenkorb" abgelegt werden können.
- Die Bestellliste kann jederzeit editiert werden.
- Beim Verlassen des Online-Shops kann die Bestellung (Verrechnung über Kreditkarte) aufgegeben werden.
- Der Inhalt des e-Katalogs wurde mit modernen Content-Management-Systemen bearbeitet und kann daher auch als DVD oder normaler Katalog (Papier) bestellt werden (gegen eine Gebühr).

E- Katalog als VO begleitendes Beispiel 2/2



- **Benutzergesteuert**
 - Grafische Benutzeroberfläche, Web-Applikation, benutzerfreundlich
 - Z.T automatisierte Vorgänge: Datenbankeintrag, Datenbankabfragen, Versenden von Bestätigungs-E-Mail
- **Sicherheit:**
 - Sichere Übertragung vertraulicher Daten (Kreditkartennummer etc.)
 - System gegen unbefugte Zugriffe schützen (intern und extern)
- **Zuverlässigkeit:** Hohe Erreichbarkeit des Systems, auch bei erhöhter Belastung
- **Wartung:** Ohne längeren Ausfall des Systems
- **Kosten:** Erschwinglicher Betrieb
- **Verwendbarkeit:** Einfach, übersichtlich für Normalverbraucher

Interaktives Beispiel



Szenario

- Der E- Katalog soll in akzeptabler Zeit entwickelt werden.
- Wir nehmen an, dass wir das Folgende ausreichend zur Verfügung haben:
 - Ressourcen (Geld, Leute, Materialien)
 - Jede nötige fachliche Kompetenz
 - Infrastruktur (Arbeitsstationen, Entwicklungsumgebung, etc.)
- Schreiben Sie 5 Probleme auf, warum das Projekt dennoch scheitern kann.
- Nennen Sie zu jedem Problem einen Lösungsansatz.

Top Ten Risikofaktoren: warum Projekte scheitern



1. Unvollständige Anforderungen
2. Anwender nicht involviert
3. Zu wenig Ressourcen
4. Unrealistische Zeit- und Kostenpläne
5. Keine Management Unterstützung
6. Häufige Änderung der Anforderungen
7. Qualitätsmängel bei extern vergebenen Komponenten
8. Qualitätsmängel bei extern vergebenen Aufgaben
9. Fehlende Planung
10. Projekt wird nicht mehr benötigt.

Faktoren für ein erfolgreiches Projekt



1. Einbringung und Berücksichtigung der **Anwender**
2. Adäquates **Projektmanagement**: nicht zu viel aber auch nicht zu wenig
3. **Anforderungen** müssen eindeutig beschrieben, realisierbar und auch überprüfbar sein
4. Flexibler, realistischer **Projektplan**, der mögl. Verzögerungen berücksichtigt
5. Realistische **Kostenschätzung** und Budget, inkl. **Risikoanalyse**
6. Angemessene Ziele
7. Schlüsselteammitglieder haben genügend **Projekterfahrung**
8. Gute **Teamarbeit**, funktionierende **Kommunikation im Team**

Zusammenfassung und Ausblick



- **Zusammenfassung**

- Verschiedene Anforderungen erfordern verschiedene Lösungsansätze
- Mit der Projektgröße ändern sich viele Faktoren des Projekts
 - Komplexität, Bedarf an Planung, Organisation etc.
- Die Eigenentwicklung von Software ist nicht immer die beste Lösung
- Unterschiede eingebettete Systeme vs. kommerzielle Software

- **Ausblick**

- Vorgehensmodelle, Vergleich einer Auswahl
- Übersicht über Produkte, was muss beachtet werden
- Verfolgen von Anforderungen

Software Engineering Vorlesung

Einheit 1-3 Software Prozesse und Produkte



Inhalt

- Vorgehensmodelle
- Produkte des V-Modells
- Produktqualitäten
- Verfolgen von Anforderungen



Schatten, Biffl, Winkler, et al. (2010) „Best Practice Software-Engineering - Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen“; Spektrum Akademischer Verlag. Kap. 3

.....

Motivation - Ziel

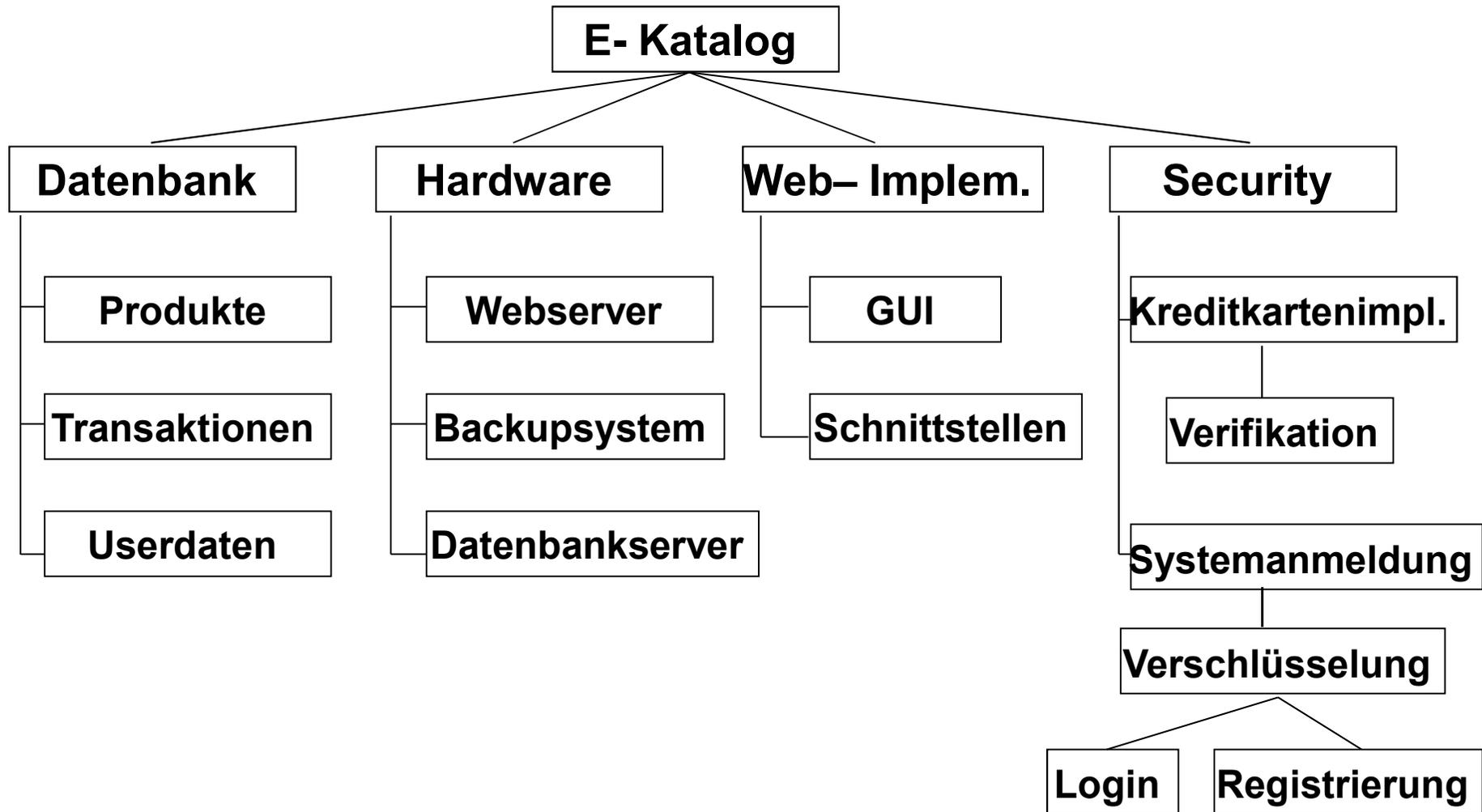


- Software-Entwicklungsprozess strukturieren
- Verbesserung der Kommunikation zwischen allen Beteiligten
- Orientierungshilfe für die Produkte
- Produkte und deren Eigenschaften
- Wann ist welches Vorgehensmodell sinnvoll

CMMI Ebenen 2 & 3 – wiederholbare Produktmerkmale, Prozesse und Methoden

.....

E-Katalog Projektmanagement Produktbaumstruktur (PBS)



Produktqualitäten 1/2



- Anforderungen
 - stabil und ausreichend genau beschrieben, übersichtliche Darstellung
 - Systembeschreibung, Anwendungsfälle, Nichtfunktionale Anforderungen
- Design
 - Skalierbarkeit, Performance, Erweiterbarkeit
 - Übereinstimmung mit Anforderungen, ausreichend dokumentiert
- Testplan
 - Testrahmen, Teststrategie, Testfälle, Testzeitplan
 - Nachvollziehbare Beschreibung, Anforderungen vollständig abgedeckt
- Testbericht
 - Version des Systems und der DB, Testfälle mit unerwartetem Ergebnis
 - verwendete Eingabedaten, welche Korrekturen sind nötig

Produktqualitäten 2/2



- Benutzerschnittstelle
 - Usability für Zielgruppen, Überprüfung mit Prototyp
 - Dialoge nicht überladen, einheitliches Layout, Fehlermeldungen einheitlich
- Datenbank
 - Redundanzfrei, Wachstum der Datenbestände berücksichtigen
 - Übereinstimmung mit Klassenmodellen, Integritätsbedingungen
- Technische Dokumentation
 - Verständlich, korrekt, vollständig, Format und Stil ansprechend
 - Lehrbuchteil, Nachschlageteil, Stichworte/Glossar, Online-Hilfe
- Projektplan
 - Alle Tätigkeiten mit Verantwortlichen und Mitarbeitern
 - Realistische Angaben für Aufwand, geeignete Darstellung

Verfolgen von Anforderungen



- Validierung der Anforderungen → beschreiben Anforderungen was der Kunde haben möchte? (Überprüfung z.B mit Prototyp)
- Anforderungen an die Anforderungen:
 - Gültigkeit ▪ Vollständigkeit ▪ Überprüfbarkeit ▪ Verfolgbarkeit
 - Konsistenz ▪ Realismus ▪ Verständlichkeit ▪ Anpassbarkeit
- Vier Richtlinien:
 - **Value-driven requirements**
Erfordert Geschäftsfall-Analyse
 - **Shared-vision driven requirements**
Stakeholder bewerten regelmässig die Anforderungen → gesamtheitliche Sicht ist wichtiger als präzise Anforderungen
 - **Change-driven requirements**
Die Erweiterbarkeit und Anpassung wird oft vernachlässigt
 - **Risk-driven requirements**
Wie detailliert müssen Anforderungen sein?
“If it’s risky to leave it out, put it in“

Boehm, B. (2000). Requirements that handle IKIWISI, COTS, and rapid change. *Computer*, 33(7), 99-102.

Siehe auch „Requirements Analysis & Specification“ VU

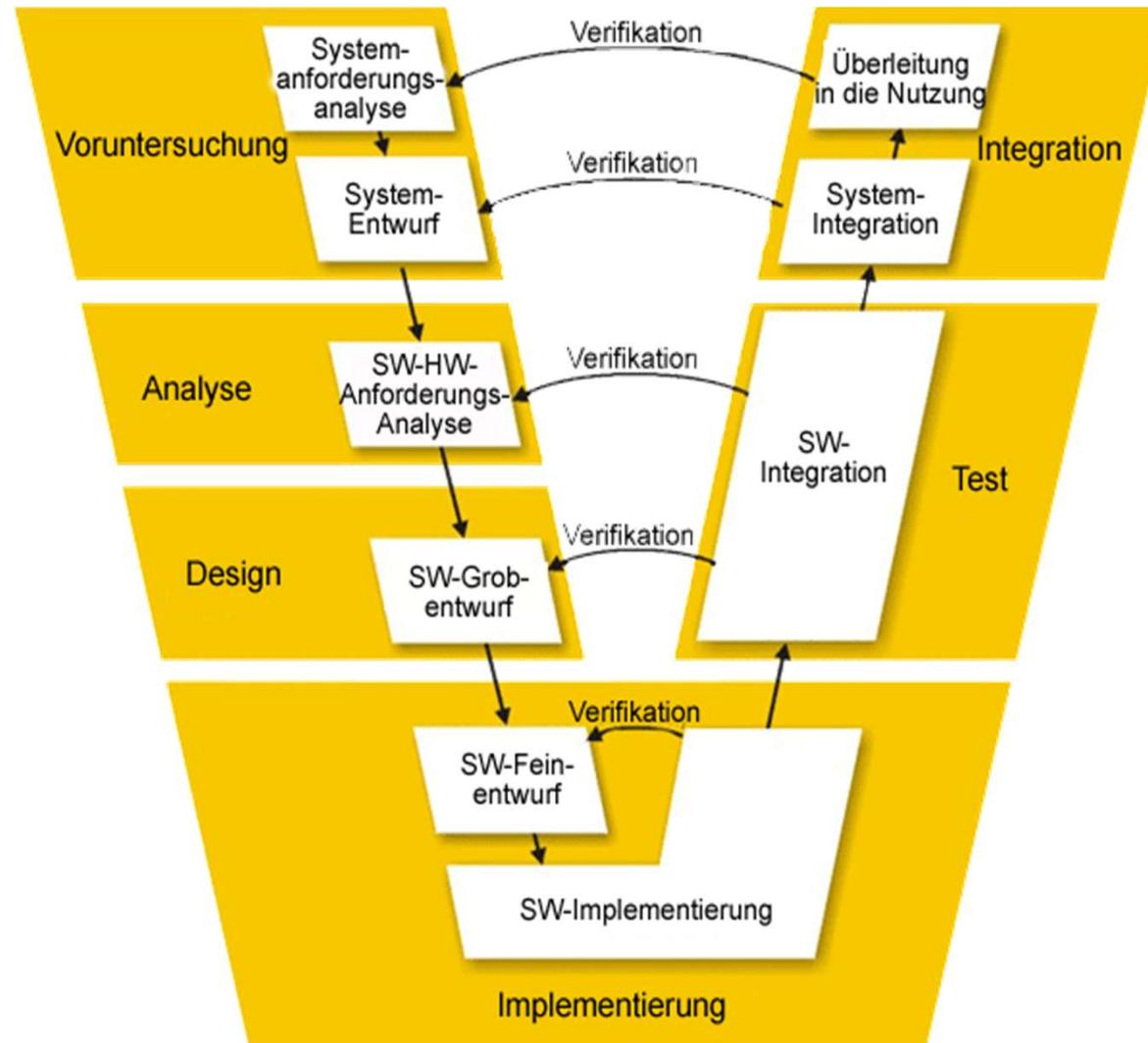
.....

Überblick und Koordination – Charakteristika einiger Prozessmodelle



Modell	Phasen	Schwerpunkte
V-Modell	Voruntersuchung, Analyse, Design, Implementierung, Test, Integration	Dokumentation, Qualitätssicherung, Verbesserung der Kommunikation aller Beteiligten
Inkrementelles Modell	Analyse, Entwurf, Implementierung, Integration, Auslieferung an Kunden	Minimale Entwicklungszeit, Risikominimierung, kurze Phasen
Iteratives Modell, z.B. Unified Process	Etablierung, Entwurf, Konstruktion, Übergang	Architekturzentriert, Anwendungsfall gesteuert
Extreme Programming (XP)	Coding, Testing, Listening, Designing laufen dauern parallel ab und nicht in Phasen	Frühe Fehlererkennung, Minimale Entwicklungszeit, Schnelle Anpassung an sich ändernde Anforderungen

V-Modell



Dröschel, W., & Wiemers, M. (Eds.). (2015). *Das V-Modell 97: der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz*. Walter de Gruyter GmbH & Co KG.

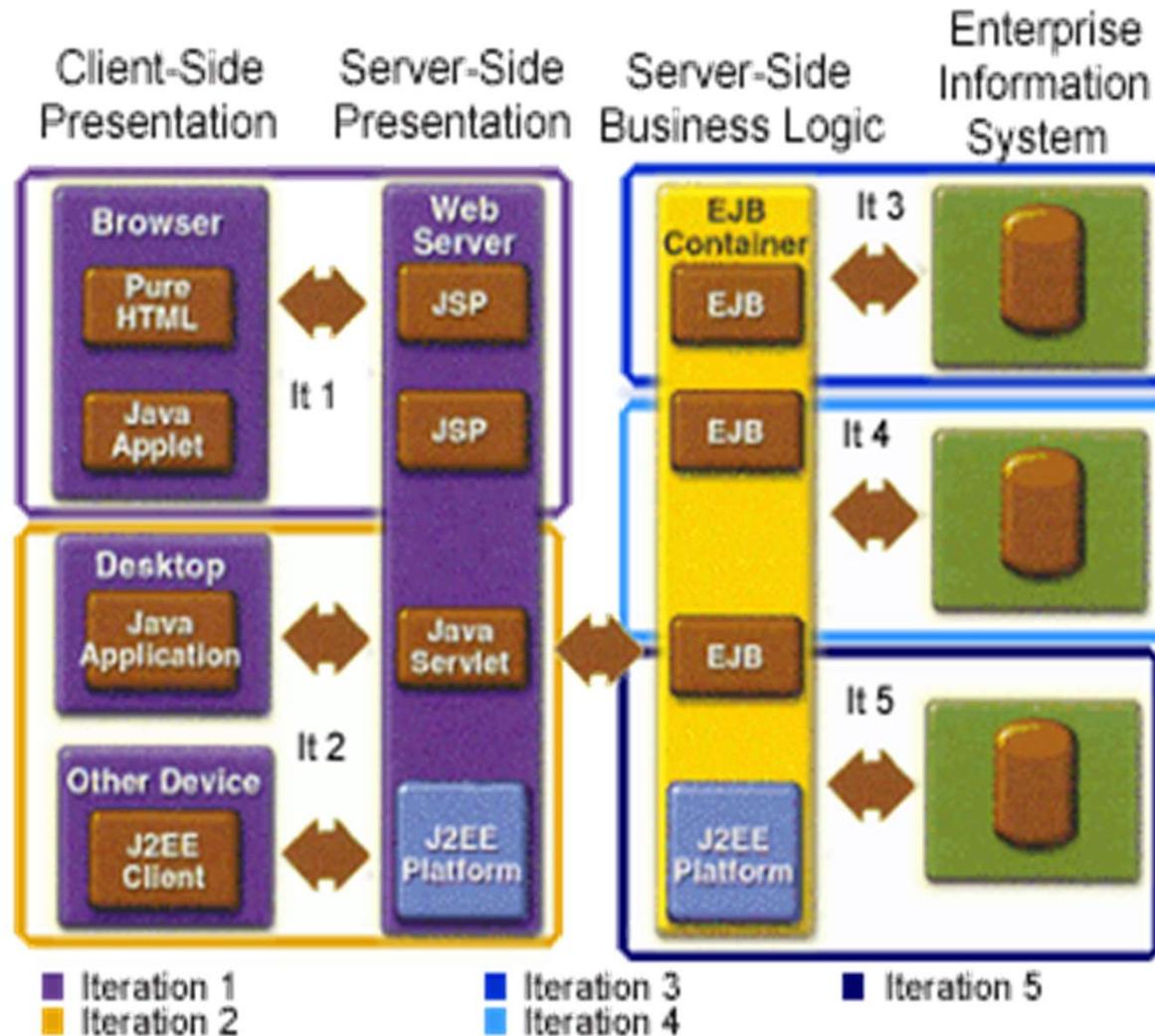
V-Modell



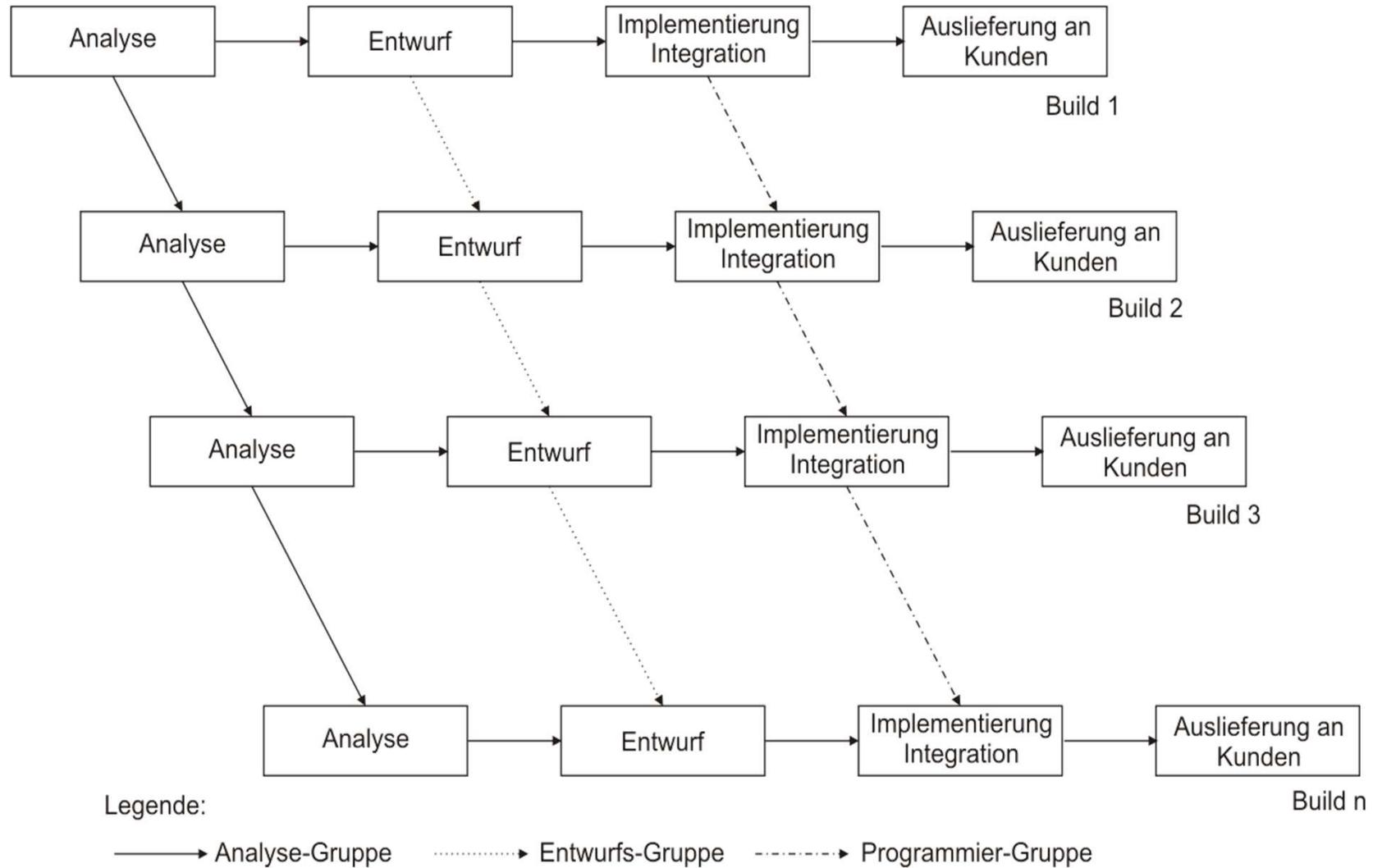
- Gesamtaufwand sinkt gemessen am Lebenszyklus einer IT-Anwendung (Kostenreduktion)
- Kürzere Einarbeitungszeiten für **neue Mitarbeiter** im Projekt
- Einfachere **Kontrolle des Projektfortschritts** (auch für den Auftraggeber)
- Projekte leicht vergleichbar → genauere **Aufwandsabschätzung** zukünftiger Projekte
- 4 Submodelle: Systemerstellung, Qualitätssicherung, Konfigurationsmanagement, Projektmanagement
- **Anwendung:** für kleine Projekte zu detailliert, für **große Projekte** mit hohem Qualitätsanspruch geeignet (speziell **Embedded Systems**)

Dröschel, W., & Wiemers, M. (Eds.). (2015). *Das V-Modell 97: der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz*. Walter de Gruyter GmbH & Co KG.

Implementierung mit Iterationen / Inkrementen



Inkrementelles Vorgehensmodell

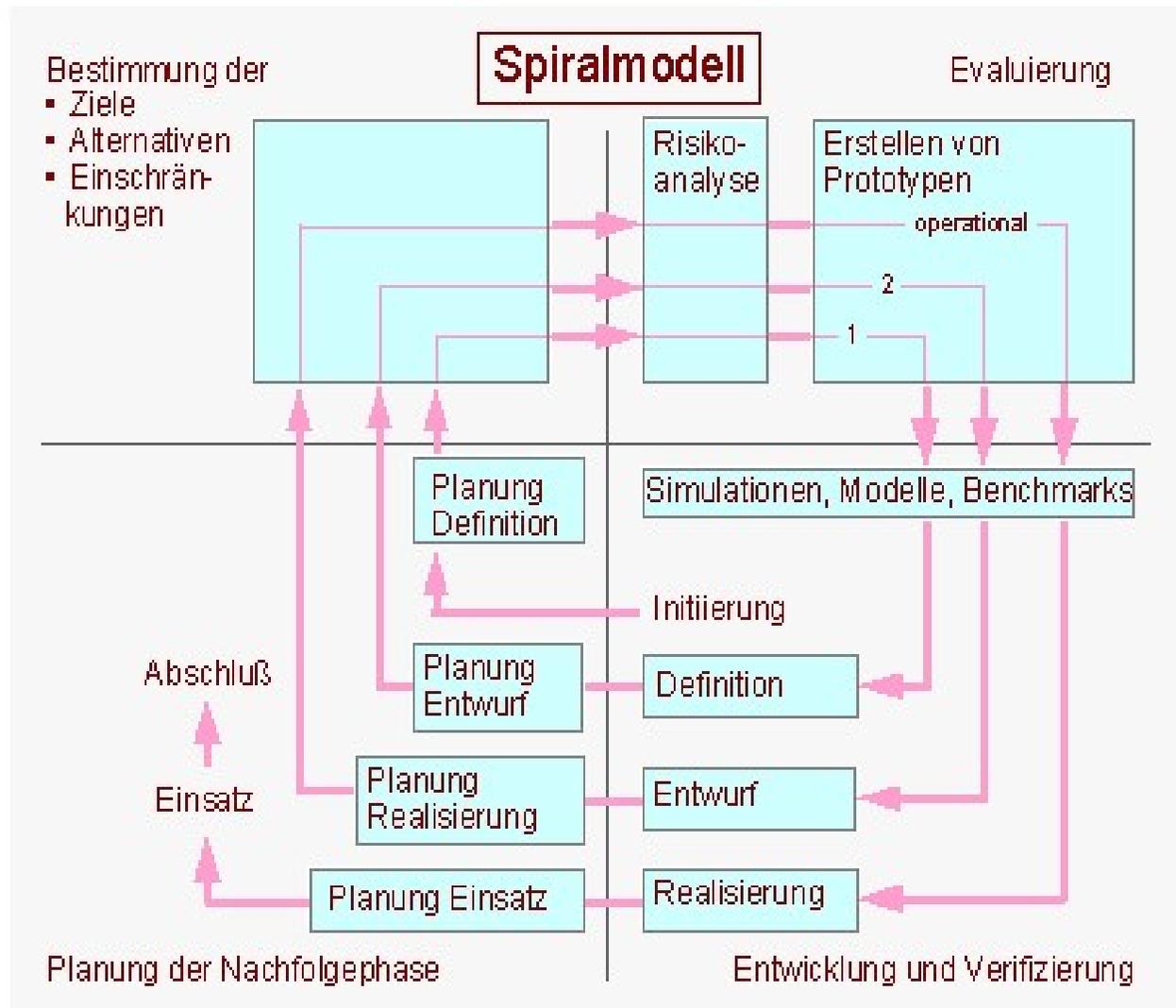


Inkrementelles Modell



- Stufenweise Entwicklung des Produkts
- Integration findet kontinuierlich statt
- Koordinierung in kleinen Etappen
- Planung wird leichter
- Planverfolgung einfacher
- Kontinuierlich hohe Qualität
- Bei unklaren Anforderungen gut geeignet
- Jeder Zyklus wird mit Meilenstein abgeschlossen
- Zyklen werden gemeinsam geplant
- **Anwendung:** große, komplexe Systeme mit langer Entwicklungszeit, wenn das Basisprodukt schnell beim Kunden sein muss, dann weiterentwickelt wird

Spiralmodell



Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61-72.

Spiralmodell



- Risikogetriebenes Vorgehensmodell
- Für jedes Teilprodukt und jede Verfeinerungsebene werden vier zyklische Schritte durchlaufen
- Ziele eines Zyklus ergeben sich aus den Ergebnissen des letzten Zyklus
- Sehr flexibles Modell
- Hoher Managementaufwand, da oft neu entschieden werden muss
- Testaufwand ist einfach einzuschätzen
- Anwendung: größere, risikoreiche Projekte, interne Softwareentwicklungen

XP: Extreme Programming



- Iteratives Vorgehensmodell, bei jeder Iteration wird mehr **Funktionalität** realisiert (sehr kurze Iterationen, **Refactoring**)
- **Coding:** kontinuierliche Integration, Programmieren in Paaren (der eine programmiert, der andere testet, Rollen können wechseln)
- **Design:** einfache Dinge nur einmal tun
- **Dokumentation:** selbstdokumentierend im Quelltext
- **Test:** Unit-Tests, Testfälle im Vornherein spezifizieren
- **Integration:** mehrmals täglich soll der Quellcode in die zentrale Code-Basis eingefügt werden, inklusive Tests
- **Anwendung:** Projekte mit schnell wechselnden oder vagen Anforderungen, mit kleinen Entwicklergruppen (bis 12 Personen), zeitkritische Projekte

<http://agilemanifesto.org/> Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, 9(8), 28-35.

Interaktives Beispiel – e-Katalog



- Es soll eine **Web-Applikation** realisiert werden, die Produkte visualisiert darstellen kann
- Produkte können in einen **Warenkorb** gelegt werden
- Der Inhalt des Warenkorbs kann jederzeit bearbeitet werden
- **Bestellung** von Produkten (**Verrechnung** über Kreditkarte)
- Der Inhalt des e-Katalogs wurde mit modernen **Content-Managementsystemen** verarbeitet und kann so bei Wunsch auch per DVD oder als normaler Katalog (Papier) bestellt werden (gegen eine Gebühr)
- Das Projektteam umfasst **15 Personen**, der e-Katalog soll ständig verbessert und **weiterentwickelt** werden
- **Welches Vorgehensmodell wäre für dieses Projekt geeignet und warum?**

Zusammenfassung



- Unterschiedliche Projekttypen erfordern die Wahl des richtigen Vorgehensmodells
- Produkte haben mehrere Qualitäten, die zu beachten sind
- Im gesamten Projekt müssen die Anforderungen immer gegenwärtig bleiben, d.h. es muss überprüft werden, ob die gewünschten Anforderungen erfüllt werden und geänderte Anforderungen müssen berücksichtigt werden.

Referenzen



- Schatten, Biffli, Winkler, et al. (2010) „**Best Practice Software-Engineering - Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen**“; Spektrum Akademischer Verlag.
- Balzert, H. (2006). *Lehrbuch der Software-Technik*. Spektrum Akad. Verlag.
- Boehm, B., & Turner, R. (2008). *Balancing agility and discipline*. 6. print., Addison-Wesley.
- Boehm, B., Lane, J. A., Koolmanojwong, S., & Turner, R. (2014). *The incremental commitment spiral model: Principles and practices for successful systems and software*. Addison-Wesley Professional.
- IEEE (1990): *IEEE Standard Glossary of Software Engineering Terminology*. IEEE STD 610.12-1990
- Kruchten, P. (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.
- IEEE (2014), *Guide to the Software Engineering Body of Knowledge*, 3rd ed., IEEE.
- Sommerville Ian (2018) „*Software Engineering*“, 10th Edition, Pearson.
- Van Vliet, H. (2008) *Software Engineering - Principles and Practice*. Wiley & Sons.
- Wallmüller, E. (2011) *Software Quality Engineering*; 3. Auflage, Hanser.

Web-Ressourcen



[QSE] <http://qse.ifs.tuwien.ac.at>

[BPSE] <http://bpse.ifs.tuwien.ac.at/>

<http://best-practice-software-engineering.blogspot.com/>

<http://bpse.ifs.tuwien.ac.at/podcast.html>

[CMMI] <http://www.sei.cmu.edu/cmami/>

[IEEE] IEEE Software Engineering Body of Knowledge (SWEBOK) www.swebok.org

[PMBok] <http://www.pmi.org/PMBOK-Guide-and-Standards.aspx>

[Pris03] TU-Chemnitz: Lehrveranstaltung: Planung und Realisierung von Informationssystemen,

http://www.tu-chemnitz.de/wirtschaft/wi1/lehre/2002_ws/pris/v/pris_v9.pdf

[Scha02] Schach, S.: Object-Oriented and Classical Software Engineering, WCB/McGraw-Hill, 2002

<http://www.mhhe.com/engcs/compsci/schach5/pps/schach5-chap05-14%5B1%5D.htm>

[Well] Wells, D.: When should Extreme Programming be Used

<http://www.extremeprogramming.org/when.html>

[Sommerville18] Videos on Software Engineering Ethics, Processes, Key activities

<https://iansommerville.com/software-engineering-book/videos/se/>