Vortragsreihe „ Software Engineering for Everyday Business“

Software Engineering & Software Prozesse

Teil 1b: Life-Cycle Phasen

Dietmar Winkler, Michael Pernkopf

Technische Universität Wien
Institut für Softwaretechnik und Interaktive Systeme

dietmar.winkler@qse.ifs.tuwien.ac.at
http://qse.ifs.tuwien.ac.at

Inhalt

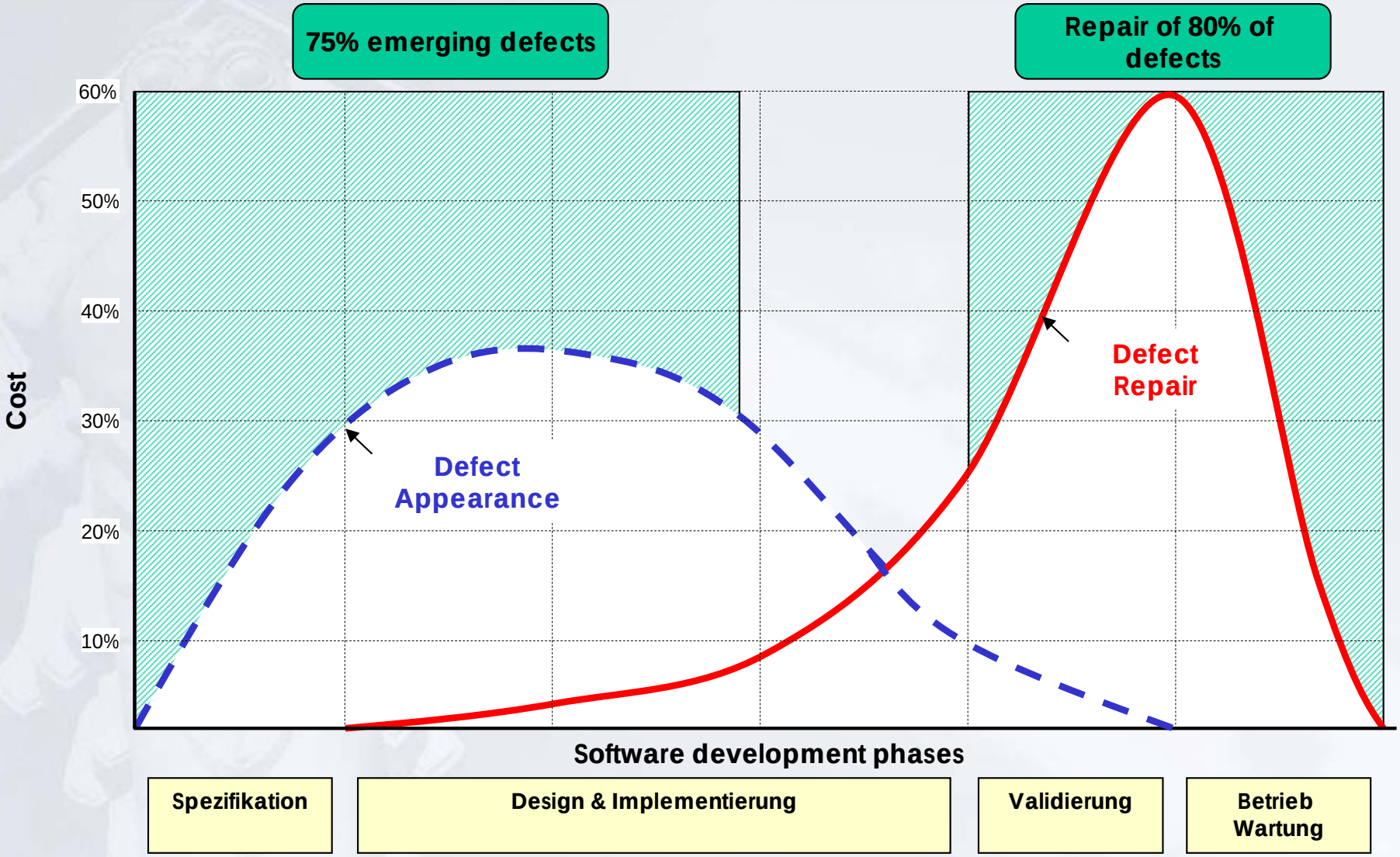
- **Teil 1a: Einführung in die Softwareentwicklung (ca. 60‘)**
  - Typisierung von Projekten
  - Erfolgs/Misserfolgsfaktoren in der Softwareentwicklung
  - Rollen in der Softwareentwicklung
- **Teil 1b: Life-Cycle Phasen (ca. 60‘)**
  - Softwarelebenszyklus
  - Ausgewählte Aspekte in den jeweiligen Phasen
  - Grundlegende Softwareprozesse in Anlehnung an den Life-Cycle Prozess.
- **Teil 1c: Ausgewählte Softwareentwicklungsprozesse (ca. 60‘)**
  - V-Modell XT
  - Rational Unified Process
  - Agile Softwareentwicklung

Motivation & Zielsetzung

- Der Produktlebenszyklus startet bei der ersten Idee und endet bei der Ausmusterung der Softwarelösung.
- Warum ist eine strukturierte Softwareentwicklung notwendig?
- Fehlerentstehung und Fehlerkosten während der Softwareentwicklung.
- Strukturierung von Software Entwicklungsprozessen.
- Anforderungen des Kunden sind die Basis für die Softwareentwicklung; durch sie wird definiert, was das Produkt leisten soll.

- Je nach Projektkriterien (Größe, Art, Typ, usw.) stehen zahlreiche Prozessmodelle zur Verfügung, die den wesentlichen Phasen des Life-Cycles folgen (Vorgehensmodelle).
- V-Modell und Inkrementelle Entwicklung als Beispiele für Vorgehensmodelle.

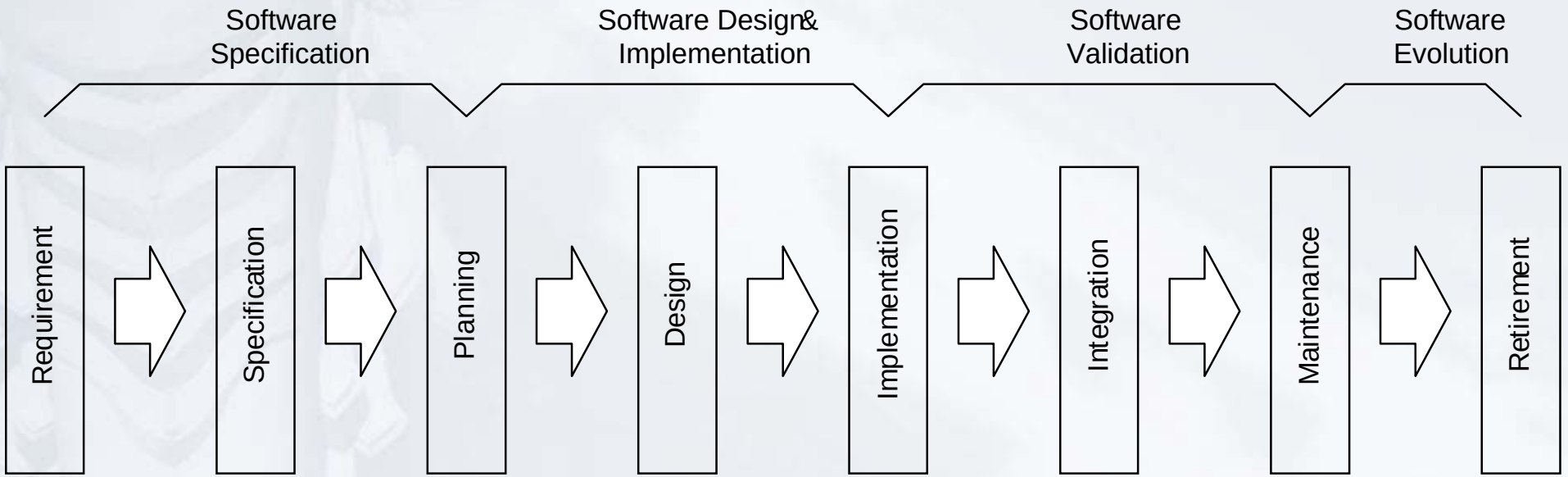
Fehler und Fehlerkosten



Source: VM XT Workshop, KMA, 2005

Übersicht über den Software Life-Cycle Prozess

- Ein Software Prozess ist eine Abfolge von Schritten (Phasen) mit all seinen Aktivitäten, Beziehungen und Ressourcen.
- Der Software Life-Cycle beschreibt ein Basiskonzept für Software Engineering Prozesse.



Software Life-Cycle Prozess

- **Requirements** (Anforderungen) zeigen die Wünsche des Kunden am Software Produkt (user/customer view).
- Eine **Specification** beschreibt das System aus technischer Sicht (*engineering view*).
- **Planning**: Erstellung des Projektplans (Zeit, Dauer, Kosten) (*project management*)
- **Design**: technische Lösung der Systemanforderungen (*Komponenten, Packaging*)
- **Implementation:** Erzeugung des Softwareprodukts (*coding/testing*).
- **Integration**: Zusammenfügen und Test der einzelnen Komponenten.
- **Operation and Maintenance**: Betrieb und Wartungsphase; Fehlerbehebung, Unterstützung, Erweiterungen des Softwareproduktes.
- **Retirement**: Nach der Einsatzphase muss das Softwareprodukt (am Ende des Produktlebenszyklus) aus dem Betrieb genommen werden.

Anforderungen

Warum sind Anforderungen so wichtig?

- Die Anforderungsanalyse ist der wichtigste Schritt in der Softwareentwicklung!
- Durch Anforderungen wird ein gemeinsames Verständnis hergestellt, was das Produkt können soll.
- Sie repräsentieren die „reale Welt“.
- Berücksichtigung unterschiedlicher Stakeholders (Kunde/Anwender, Entwickler, usw.)
  - Unterschiedliche Betrachtungsweisen des Projektes
  - Unterschiedliches Begriffsbild und Vokabular.
  - Die Projektbeteiligten müssen sich auf eine gemeinsame Sicht einigen.
- Typischerweise werden Anforderungen beschrieben bzw. grafisch dargestellt (z.B. Use-Case Modellierung aus der UML Familie)

→ **Eine Anforderung ist Ausdruck für ein gewünschtes Verhalten aus Nutzersicht**

→ **Anforderungsmanagement als Mittel zur Bewältigung der Änderungen**

Arten von Anforderungen

- **Funktionale Anforderungen**
  - Was bzw. welche Funktion soll umgesetzt werden (System Services)?
  - Wie soll sich das System in definierten Situationen verhalten?
- **Nicht-funktionale Anforderungen**
  - Welche sonstigen Anforderungen müssen umgesetzt werden, die nicht direkt auf die Funktionalität abzielen.
    Beispiele
    - Leistung und Performance (z.B. Optimierung des Informationsflusses).
    - Usability und menschliche Faktoren (z.B. Einfachheit der Bedienung, Training).
    - Sicherheit (z.B. Zugangskontrolle, Trennung von Anwendung und Daten).
- **Designbedingungen**
  - Worauf soll beim technischen Entwurf geachtet werden, z.B. Schnittstellen, Unterstützung von festgelegten Plattformen?
- **Prozessbedingungen**
  - Rahmenbedingungen im Softwareprojekt, z.B. Ressourcen / Dokumentation

Typische Probleme bei Anforderungen…

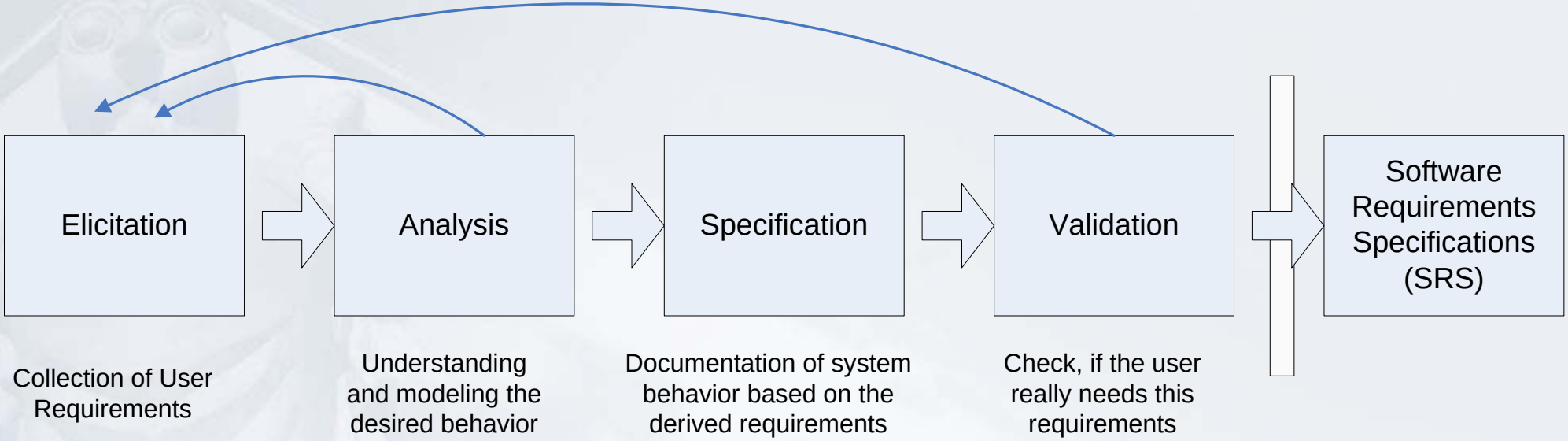
ANWENDER (USER)

- .. Wissen nicht was sie wollen, sie haben nur eine präzise Idee …
- .. Nicht kompromissfähig ..
- .. Lehnen es ab Verantwortlichkeiten für das System zu übernehmen ..
- .. Wollen das gesamte Paket sofort und zu einem niedrigen Preis..
- .. Bedürfnisse schwer priorisierbar ..
- .. Üblicherweise nicht interessiert in Technik ..

ENTWICKLER

- … üblicherweise kein Wissen über die Domäne …
- .. 'Nein'-Sager ..
- .. Versuchen dem User seine Arbeit zu beschreiben ...
- .. Immer spät dran und verschwenden Zeit sowie Budget ..
- .. Fokus auf Technik nicht auf wirtschaftlichen Faktoren …

Anforderungsprozess



- Erhebung der Anforderungen aller relevanten Stakeholder ist zeitaufwendig und nicht trivial.
- Priorisierung von Anforderungen:
  - Hauptanforderungen (must-be Kritieren),
  - Gewünschte Anforderungen (nicht entscheidend aber wichtig)
  - Optionale Anforderungen (nice-to-have Kriterien).

Spezifikation / Planung

Spezifikation

- Vorbedingung: Anforderungen aus Kundensicht und Projektziele sind klar.
- Detailinformation über funktionale und nicht-funktionale Anforderungen (einfach, z.B. auch für Nicht-Techniker verständlich).
- Projektumgebung: Systemumgebung, Testszenarien (Anwendungsenbene), Anforderungen an die Dokumentation (Benutzerhandbuch, Online-Hilfe) etc.
- Wichtigstes Dokument für die weitere Softwareentwicklung und Ausgangsbasis für die weiteren Schritte.
- Die Spezifikation ist in den meisten Fällen auch die Grundlage für die Vertragsbildung zwischen Auftraggeber und Auftragnehmer.

Planung:

- Detaillierte Planung des weiteren Projektes (Definition der Vorgehensweise, Projektplan mit Meilensteinen, Iterationen, Testpläne,..)

Software Design & Implementierung

Software Design:

- Dieser Schritt beinhaltet neben der konkreten Vorstellung, was das Produkt leisten soll, auch Information, wie diese „Leistung“ umgesetzt wird.
- Das Software Design beinhaltet technische Anwendungsfälle der Anforderungen, alle ihrer Subsysteme (Komponenten) und Detailinformation, wie die Lösung aus technischer Sicht aussehen soll.
- Softwaredesign enthält interne Datenstrukturen, Datenflüsse, Algorithmen,..
- Sie ist in der Regel in der „Sprache der Techniker“ geschrieben.
- Das Designdokument ist die Grundlage der Implementierung!

Implementierung / Umsetzung:

- Programmieren und Testen der einzelnen Komponenten sowie einfache Integrationstests (einzelne Module zu einem größeren Ganzen kombinieren).

Wichtige Punkte bei der Implementierung

- Neben der technischen Umsetzung der Problemlösung (basierend auf dem Design) müssen auch “formale” Kriterien berücksichtigt werden, um qualitativ hochwertige Produkte zu erstellen.
- Beispiele:
  - Lesbarkeit und Verständlichkeit des Softwarecodes um das Produkt besser testen, warten, wieder verwenden und weiterentwickeln zu können.
  - Verwendung von Dokumentation
    - innerhalb des Programms und
    - in externen Dokumenten
      um den Code zu beschreiben (Codeorganisation, Daten, Funktionen,…).
  - Objektorientierung als Grundlage für unabhängige Komponenten (Objekte).
  - Standardisierung in der Softwareentwicklung (z.B. SWEBOK, Industriestandards).
    - Einheitliche Namensgebung von Modulen, Variablen, Funktionen
    - Code Formatierungen (z.B. Einrückungen)
    - Versionskontrolle (z.B. CVS) usw.

Verifikation und Validierung

- Fehler im Softwaredesign haben oft immense Auswirkungen auf die Produktqualität, Projektdauer und Projektbudget.
- Einsatz an Nacharbeit steigt je später ein Fehler gefunden wird.
- Ziel ist es daher, Fehler möglichst frühzeitig zu erkennen und zu beheben.
  - Ein Fehler ist dabei eine Abweichung zwischen Lösung (Komponente, Modul,..) und seiner Spezifikation bzw. der erwarteten Eigenschaft.

- Unter Verifikation versteht man die Prüfung eines Produkte im Hinblick auf die Spezifikation. „Wurde das Produkt richtig entwickelt“.
  z.B. Modul- und Komponententests (Prüfung der Lösung gegen die techn. Spezifikation)
- Im Rahmen einer Validierung wird die Frage beantwortet, ob das richtige Produkt entwickelt wurde (also der Kunde auch das bekommt, was er bekommen wollte).
  z.B. Akzeptanz- und Abnahmetests (Prüfung der Lösung gegen die Anforderungen)

Grundlegende Testtechniken

- **Modul / Unit Test**:
  Fokus auf Basiskomponenten (Module, Softwarefragmente) und Übereinstimmung zwischen Komponenten und technischer Spezifikation.
- **Integrationstest**:
  Fokus auf Interfaces und Verbindungen zwischen Komponenten innerhalb des (Sub)Systems.
- **Systemtest**:
  Fokus auf Übereinstimmung zwischen funktionalen und technischen Bedingungen.
- **Regressionstest**:
  Testen von modifizierten Funktionen (Fehler durch Modifikationen verhindern).
- **Akzeptanztest**:
  Übereinstimmung der festgelegten Anforderungen.
- **Installationstest**:
  Identifizieren von Fehlern während der Installation.

Wichtige Punkte beim Testen

- Entwickler sind verantwortlich für die Entwicklung von Softwareprodukten
  → NICHT die Tester!
- Entwickler können/sollen den eigenen Code nicht testen („4-Augen-Prinzip“).
  → Rollenverteilung innerhalb des Softwareprojekts.

- Ziel des Testens ist es, Fehler im Softwareprodukt zu finden
  → und NICHT Beseitigung der Fehler!
- Test ist eine Kernaktivität,
  → keine Zusatzaufgabe am Ende des Projekts.
- Testprozesse erfordern ebenfalls Aufzeichnungen zur Wiederholung der Tests, zur Fehlerbehebung von erkannten Fehlern, usw.

Integration- und Wartungsphase

Software Integration

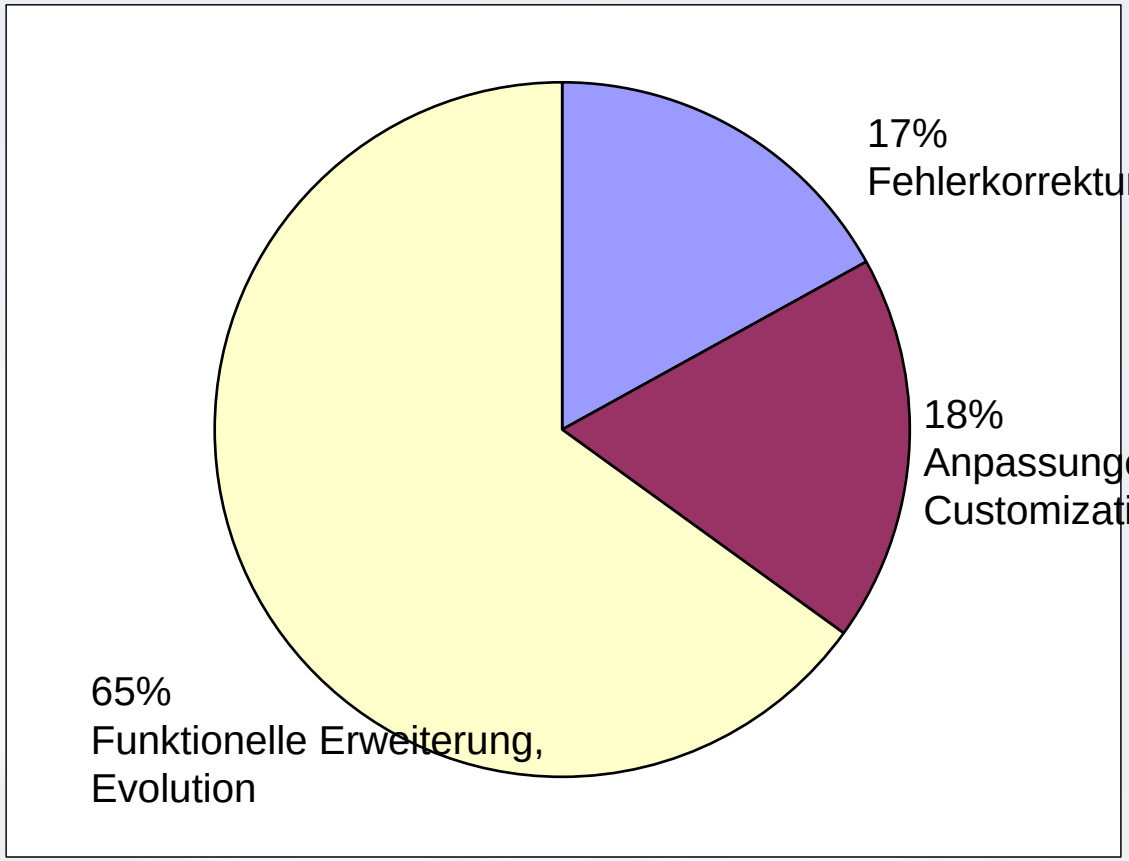
- Integration hilft bei der Konstruktion von (Sub)Systemen basierend auf Einheiten und Modulen.
- Integration beschreibt das „Zusammenfügen“ von einzelnen umgesetzten Komponenten entsprechend den Iterationsplänen nach definierten Strategien z.B. Big-Bang Integration, Top-Down und Bottom-Up Strategie, Build-Integration
- Ausführen von Iterationstest entsprechend den definierten Testplänen.

Wartungsphase

- ist ein Teil des Life-Cycle Prozesses, während sich die Lösung bereits im Einsatz befindet.
- Wesentliche Unterscheidung zwischen
  - Korrektiven Wartungsaktivitäten, z.B. Korrektur von Fehlern
  - Nicht-Korrektiven Wartungsaktivitäten, z.B. Erweiterung / Anpassungen des Softwareproduktes

Verteilung des Wartungsaufwands

- Wartung verbraucht einen großen Teil der finanziellen Ressourcen.
- Mehr als 80 % der Wartungs-arbeiten werden für nicht-korrektive Tätigkeiten benutzt.
- Wartungskosten hängen ab von
  - Applikationstyp.
  - Neuheit des Produktes.
  - Verfügbarkeit von Personal.
  - Hardwarecharakteristika.
  - Qualität des Softwaredesigns, Konstruktion, Dokumentation und Testen



Kosten von Software Wartung [Sommerville, 2001]

Wartungsprozesse beinhalten dieselben Phasen wie der Life-Cycle Prozess; einen speziellen Stellenwert nehmen die Anforderungsänderungen ein.

Retirement Phase

Retirement

- Am Ende der Operation Phase (Betrieb und Wartung) schließt der Life-Cycle Prozess mit der „Ausmusterungsphase“ ab.
- Kontrolliertes Außer-Betrieb-Setzen des Produktes.

Gründe für Ausmusterung:

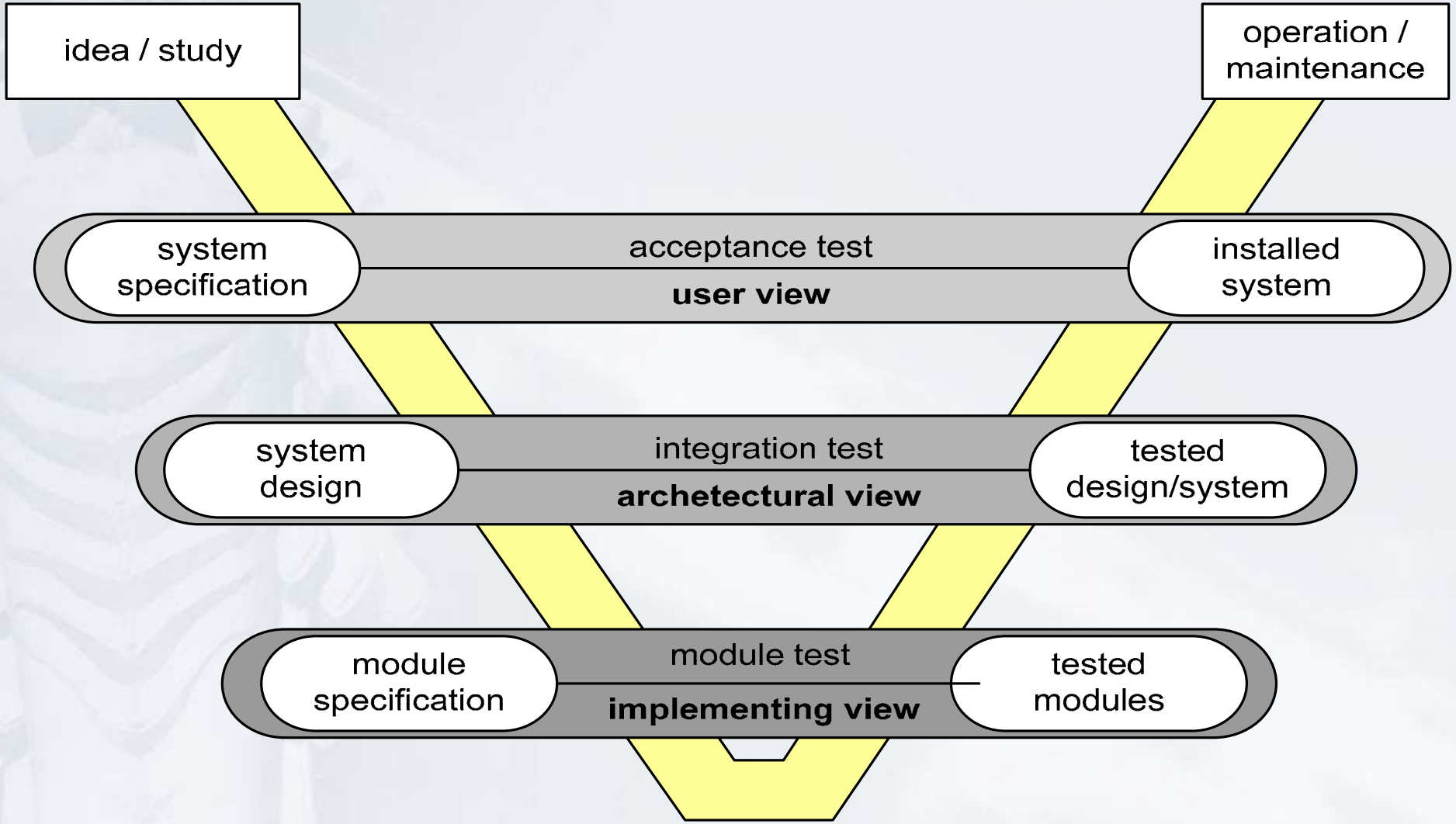
- Zahlreiche Änderungen während der Wartungsphase können ein komplettes Redesign des Produktes verursachen.
- Laufzeitfehler durch Nebeneffekte (kleine Änderungen im Programmcode können große Auswirkungen im Programmablauf haben).
- Inkonsistenzen durch “quick and dirty” Änderungen ohne Aktualisierung der dazugehörigen Dokumentation.
- Hardwareänderungen können ebenso ein komplettes Redesign oder Neuprogrammierung verursachen.

Vom Software-Life-Cycle zum Prozessmodell

- Die grundlegenden Phasen des Life-Cycle Prozesse finden sich in allen Produkten und Projekten.
- Der Schwerpunkt der meisten Vorgehensmodelle liegt auf der technischen Seite, sie beginnen bei der Definition der Anforderungen und enden bei der Inbetriebnahme beim Kunden.
- In der Praxis finden wir eine Vielzahl von unterschiedlichen Prozessmodellen
  - Standardisierte “common” Prozessmodelle (V-Modell, RUP, Agile Ansätze)
  - Unternehmensspezifische Vorgehensmodelle, die an die Bedürfnisse der jeweiligen Unternehmen bzw. Projekte angepasst werden.

- Software Prozesse oder Vorgehensmodelle sind auf bestimmte Kriterien zugeschnitten und können – je nach Projektkontext – sinnvoll eingesetzt werden.

V-Modell Konzept



Vor- und Nachteile des V-Modell Konzeptes

Vorteile

- Spezifikationsphase vs. Realisierung und Testen.
- Kontext von Produkten und Tests.
- Verschiedene Abstraktionslevels (User, Architekten und Implementierungssicht).
- Fehlerbehandlung in frühen Phasen des Softwareentwicklung (durch Einsatz von Reviews).
- Basiskonzept für VM 97 und VM XT (siehe Vortrag 1c).

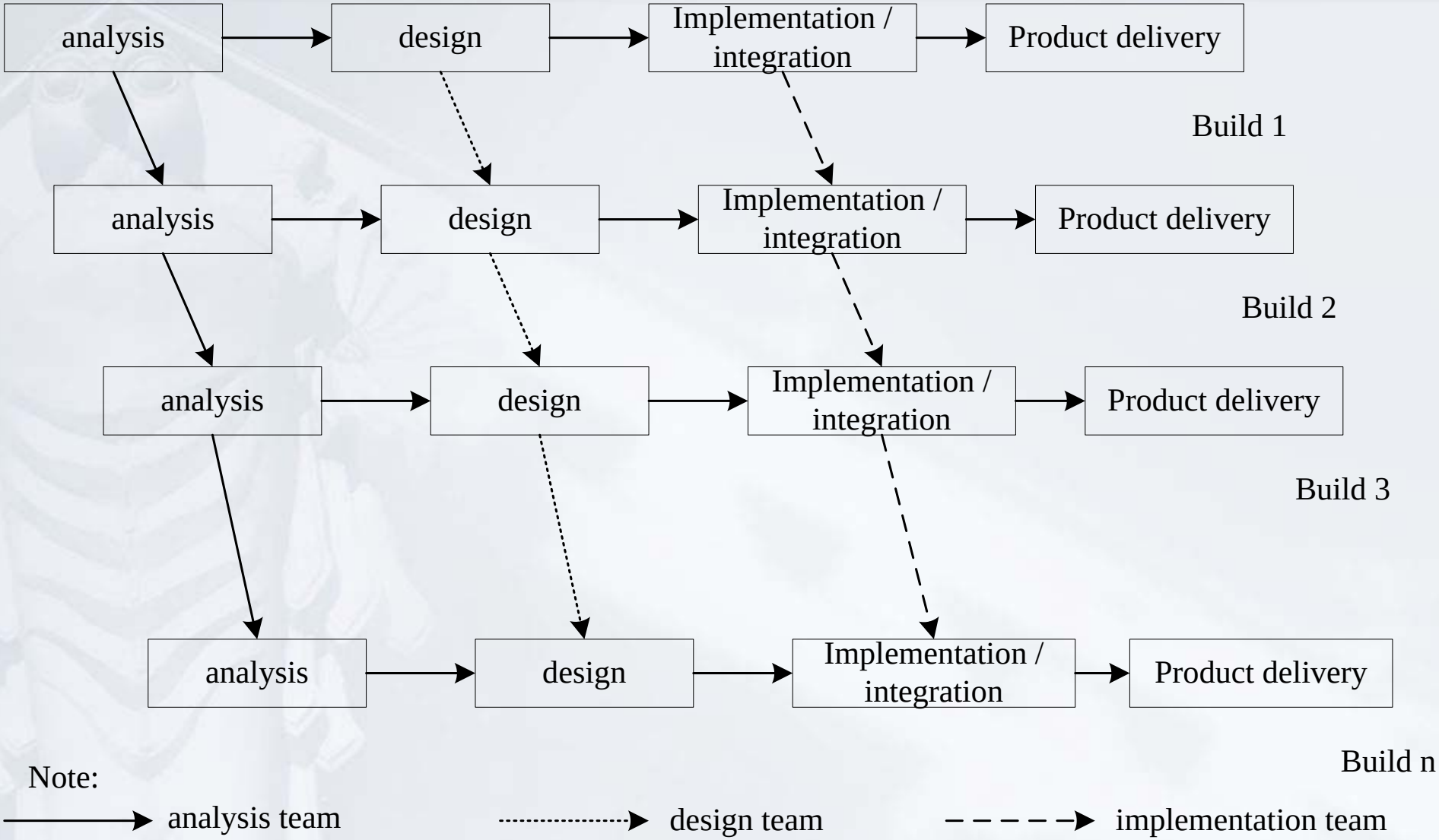
Nachteile

- Klare Beschreibung der Systemanforderungen ist wichtig.
- Dokumentationslastig

Applikation:

- Große Projekte im öffentlichen Bereich mit klar definierten Anforderungen

Inkrementelle Softwareentwicklung



Vor- und Nachteile der Inkr. Entwicklung

- Schrittweise Produktentwicklung (verschiedene Releases, Builds).
- Fortlaufende Systemintegration.
- Kleine Schritte der Entwicklung (Softwareinkremente planen).
- Iterationsplanung enthält Meilensteindefinition nach jedem Entwicklungszyklus.

Vorteile

- Anwendbar auf unklare Anforderungen.
- Große und komplexe Systeme.
- Lange Entwicklungszeit.
- Schnelle Lieferung von Softwareteilen.

Nachteile

- Probleme, wenn Releases nicht zusammenpassen.

Applikation:

- Große, komplexe Systeme; Projekte mit langer Laufzeit.

Zusammenfassung

- Der Produktlebenszyklus eines Softwareproduktes beginnt bei der ersten Idee und Definition der Anforderungen und endet mit dem Auslauf des Produktes.
- Anforderungen werden durch den Kunden vorgegeben und müssen technisch umgesetzt werden (gemeinsame Sprache und Verständnis zwischen Auftraggeber und Auftragnehmer ist sehr wichtig).
- Fehler, die in frühen Phasen der Entwicklung gemacht und erst spät erkannt werden, können erhebliche Zeitverzögerungen und Kosten zu deren Behebung verursachen.

- Software Prozesse unterstützen die Erstellung qualitativ hochwertiger Produkte.
- Die Verwendung von Softwareprozessen garantiert aber nicht, dass auch qualitativ hochwertige Produkte erstellt werden – sie erhöht aber die Wahrscheinlichkeit dafür!
- Für unterschiedliche Anwendungen existieren zahlreiche passende Vorgehensmodelle.

Referenzen

- Balzert H.: Lehrbuch der Software-Technik, Heidelberg: Spektrum Akademischer Verlag, 1998:
- Boehm B.: „Software Engineering is a Value-Based Contact Sport“, IEEE 2002.
- Kruchten Philippe, "The Rational Unified Process, An Introduction, Second Edition", Addison Wesley, 2000:
- Shari Lawrence Pfleeger; Joanne M. Atlee, Software Engineering – Theory and Practice, Third Edition 2006, Pearson Education.
- Schach, S.: Object-Oriented and Classical Software Engineering, WCB/McGraw-Hill, 2002. http://www.mhhe.com/engcs/compsci/schach5/
- Sommerville, Ian: „Software Engineering“, 6th Edition, Addison Wesley, 2001, ISBN: 020139815x.
- SWEBOK: Guide to the Software Engineering Body of Knowledge, http://www.swebok.org, 2005.
- Hans Van Vliet, Software Engineering – Principles and Practice, Second Edition 2000, John Wiley & Sons, LTD.

Vielen Dank für die Aufmerksamkeit !

Kontakt: Dipl.-Ing. Dietmar Winkler, Michael Pernkopf

Technische Universität Wien
Institut für Softwaretechnik und Interaktive Systeme
Favoritenstr. 11/188, A-1040 Vienna, Austria

dietmar.winkler@qse.ifs.tuwien.ac.at
http://qse.ifs.tuwien.ac.at