Systementwurf

Von Sebastian Meszaros

Motivation

Vorgehensweise

Grobentwurf

Feinentwurf

Implementierung

Fazit

Motivation

- 31% aller Softwareprojekte werden als gescheitert abgebrochen

  Quelle: The Standish Group

- 52% aller Softwareprojekte kosten dreimal soviel wie ursprünglich geschätzt

  Quelle: The Standish Group

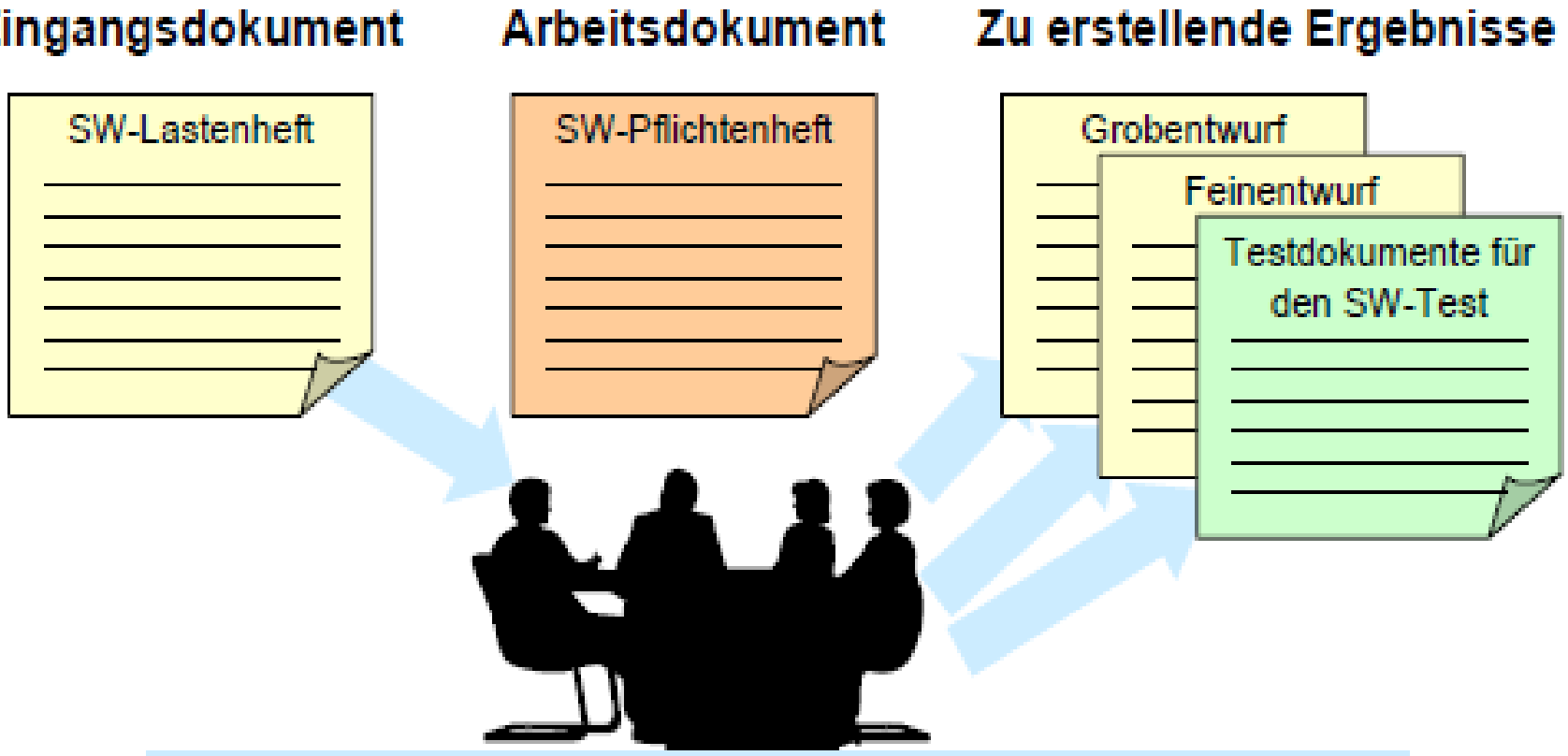
- große Softwaresysteme werden im Schnitt ein Jahr zu spät ausgeliefert

  Quelle: The Standish Group

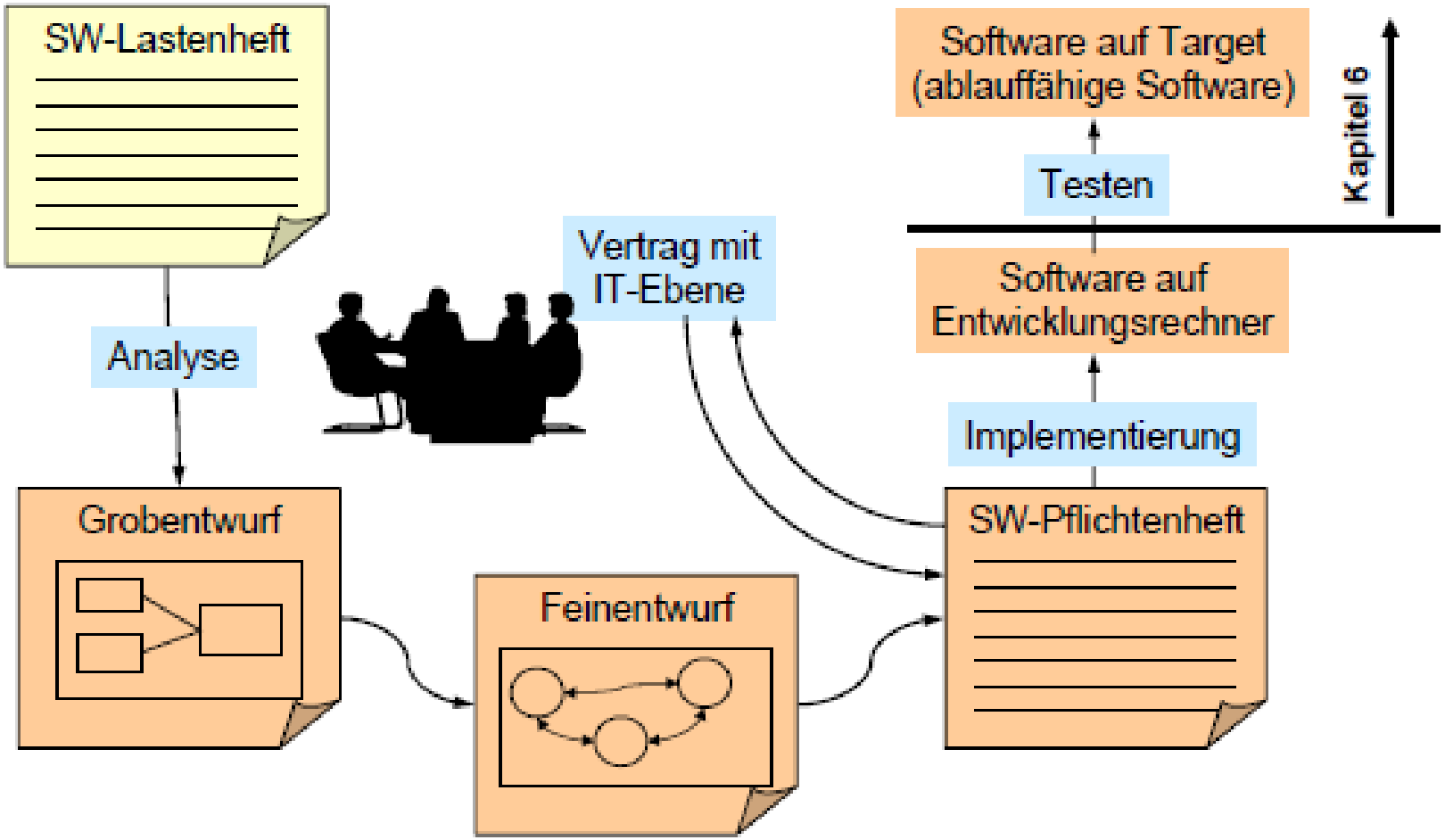
- nur 9% aller Softwareprojekte werden rechtzeitig, ohne Budgetüberschreitung und zur Kundenzufriedenheit abgeschlossen

  Quelle: T. Capers Jones

Vorgehensweise



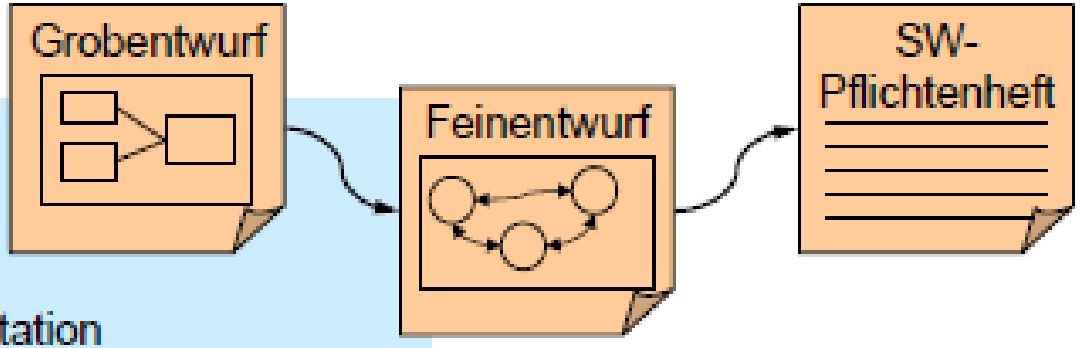
Vorgehensweise und Ziel



Analyse – zu klärende Fragen

- Wer sind die Benutzer?
- Was sind die funktionalen Anforderungen?
- Was sind die nichtfunktionalen Anforderungen?
- Verteilung? Auf welcher Steuerung läuft welche Softwarekomponente?
- Welche Softwarekomponenten kann man aus früheren Projekten übernehmen?
- Soll ein Echtzeitbetriebssystem eingesetzt werden? Wenn ja, welches?
- Welches Programmierparadigma soll verfolgt werden?
- Welche Programmiersprache soll verwendet werden?
- Welche Entwicklungsumgebungen stehen zur Verfügung?

SW-Pflichtenheft



- Produktdaten
- Produktleistungen
- Qualitätsmerkmale
- Umfang der Dokumentation
- Benutzerführung
- Gestaltung des graphischen Userinterfaces

Das Softwarepflichtenheft ist eine verbindliches Dokument und bildet die Vertragsgrundlage zwischen der IT und der SW-Ebene.

Ziel des Grobentwurfs

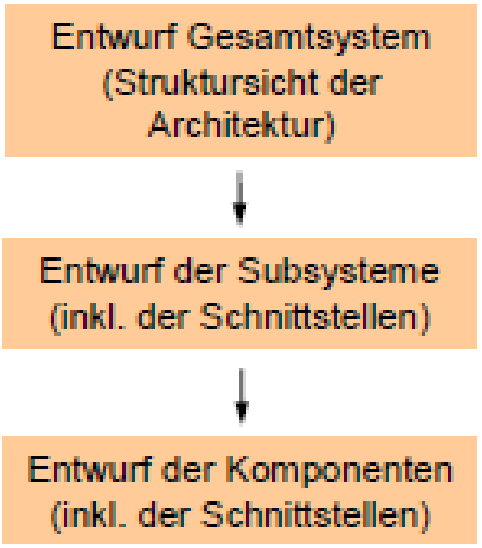
Software-Grobentwurf

- Arbeitsgrundlage für den Grobentwurf ist das SW-Lastenheft
- Beschreibung „WIE“ die Anforderungen aus dem Lastenheft erfüllt werden sollen
- Beschreibt die Architektur (Gesamtstruktur) des Software-Systems
  - Subsystem-Spezifikation
  - Schnittstellen-Spezifikation
  - Funktionale Untergliederung des Gesamtsystems

Der Grobentwurf ist prinzipiell unabhängig von der Implementierungssprache!

Vorgehen beim Grobentwurf

- Vom Groben zum Feinen (Top Down)
- Aufteilen der Funktionalität



zuerst das **Gesamtsystem**, dann die **Subsysteme** und die einzelnen **Komponenten inklusive ihrer Schnittstellen**.

Kriterien für einen „guten“ Entwurf

- Korrektheit
  - Erfüllung der Anforderungen
  - Wiedergabe aller Funktionen des Systemmodells
  - Sicherstellung der nichtfunktionalen Anforderungen
- Verständlichkeit
- Anpassbarkeit
- Hohe Kohäsion innerhalb der Komponenten
- Schwache Kopplung der Komponenten
- Wiederverwendung

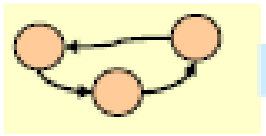
Diese Kriterien gelten auf allen Ebenen des Entwurfs (Architektur, Komponenten)!

Software-Feinentwurf

- Arbeitsgrundlage für den Feinentwurf ist der aus dem SW-Lastenheft entstandene Grob-Entwurf
- Beschreibung „WIE" die Anforderungen aus dem Lastenheft erfüllt werden sollen
- Der Feinentwurf beschreibt die Detailstruktur, sowie
  - die Logische Sicht,
  - die Verhaltenssicht und
  - die physikalische Sicht der Software

Ziele des SW-Feinentwurfs

- Dient als Grundlage für Programmierer
- Darf keinen Interpretationsspielraum lassen



```
void main ()
{
  int a_low = 5;
  int a_high = 10;
  int x = 0;
  x = get_a ();
  ...
}
```

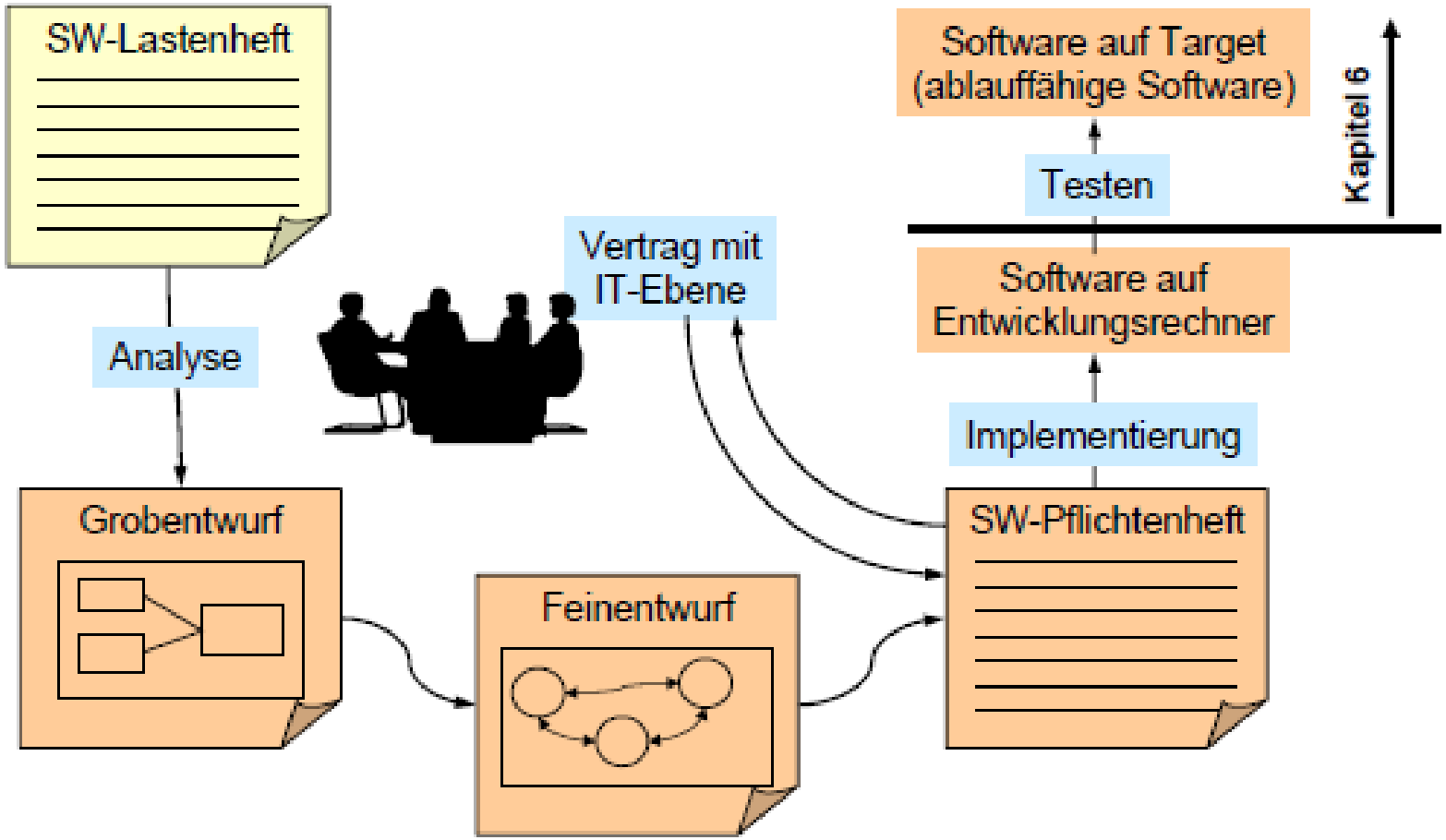
Der Feinentwurf ist angepasst an die Implementierungssprache und Plattform!

Vier Sichten der Softwarearchitektur

Vier Sichten der Softwarearchitektur – Verfeinerung und Erweiterung des Grobentwurfs (wie bei Standard SW-Entwicklung)

⇨ **Fragestellungen, die speziell bei der SW-Entwicklung für eingebettete Systeme auftreten**

Implementierung



Vom Feinentwurf zur Implementierung

Feinentwurf

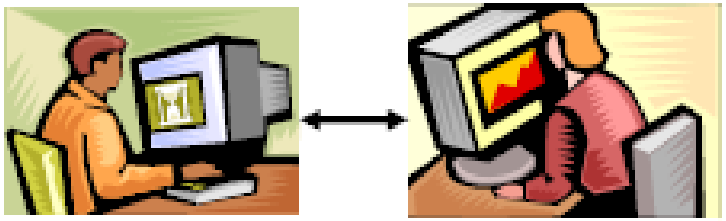
Detaillierungsgrad des Feinentwurfs muss so hoch sein, dass kein Interpretationsspielraum bei der Implementierung besteht!

Offene Fragen bei der Implementierung ⇨ Überarbeitung des Feinentwurf

Unterschiede zur Standard-SW-Entwicklung

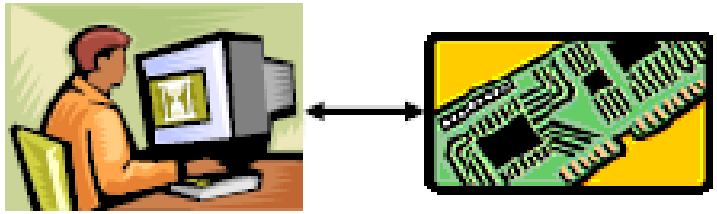
Standard Software

- Entwicklungs- und Zielplattform sind identisch → Umfangreiche Debugmöglichkeiten
- Fast keine unerwarteten Fehler
- Keine großen Unterschiede zwischen den Zielplattformen, Standardisierte Schnittstellen und Kommunikationsformen (z.B. PCI, USB, TCP/IP, ...)

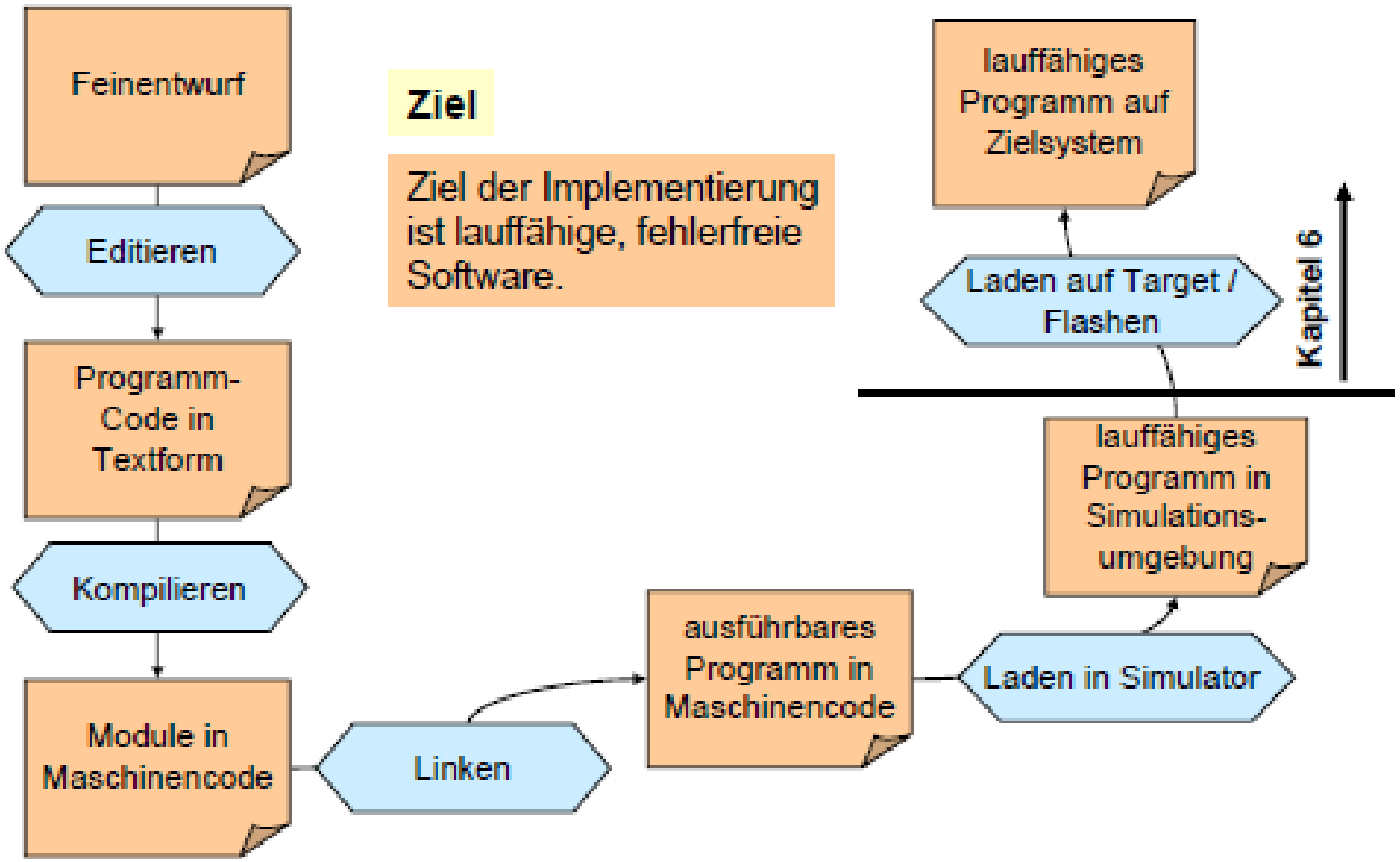


Software für eingebettete Systeme

- Entwicklungs- und Zielplattform sind unterschiedlich → Besondere Maßnahmen für Test und Inbetriebnahme notwendig
- Unerwartete Fehler
- Große Unterschiede zwischen den Zielplattformen. Viele unterschiedliche Schnittstellen und Kommunikationsformen.



Vorgehen bei der Implementierung



Ziel

Ziel der Implementierung ist lauffähige, fehlerfreie Software.

Programmierrichtlinien

Gründe für Programmierrichtlinien

- 80% der Softwareentwicklungskosten sind Wartungskosten
- Mehrere Entwickler programmieren eine Software ⇨ verteilte Entwicklung (teilweise in verschiedenen Ländern)

⇨ **Programmierrichtlinien leisten einen Beitrag zur Problembeherrschung**

Ziele von Programmierrichtlinien

- Die Lesbarkeit von Quelltexten verbessern (Insbesondere für andere Personen)
- Den Austausch von Quellen erleichtern (z.B. zwischen verschiedenen Mitgliedern eines Entwicklerteams)
- Die Qualität der Software steigern

Beispiel Entwicklungshandbuch – Richtlinien

1. Programmtexte werden in englischer Sprache geschrieben
2. Kommentare werden in englischer Sprache geschrieben
3. Schreibweise

3.1 Wortanfang

- Klassen und Typen fangen mit Großbuchstaben an
- Funktionen fangen mit Großbuchstaben an
- Variablen fangen mit Kleinbuchstaben an

Die Trennung zwischen Teilworten geschieht durch Großbuchstaben
Beispiel: setDefaultWindowColor

3.2 Keine Unterstriche

In Bezeichnern werden keine Unterstriche verwendet.

3.3 Funktionsnamen

Für Funktionen ohne Rückgabewert (C: void-Funktionen) wird als Anfang ein Verb im Imperativ verwendet. *Beispiel*: deleteListItems()
Für alle anderen Funktionen richtet sich der Name nach dem Typ des Rückgabewertes.

Versionsmanagement

Versionskontrolle

Periodisches Sichern und Archivieren eines Entwicklungsstandes

Ziele

- Qualitätssicherungsmaßnahme
- Ermöglichen einer parallelen und verteilten SW-Entwicklung
- Überblick behalten
- Verkürzung der Entwicklungszeiten
- Erhöhung der Sicherheit ("Noch die kleine Änderung, dann ...")
- „Einfrieren" bestimmter Versionen, wichtig bei Veränderung der Hardware
- Welcher SW-Stand ist auf dem Steuergerät?

Versionsmanagement

Versionsverwaltungstools

- Dokumentieren die Veränderung der Quellen
- Unterstützen Versionsmanagement
- Ermöglichen und unterstützen die Verwaltung von Codelines

Tools ersetzen nicht

- Kommunikation zwischen den Entwicklern
- Mangelnde Qualität (z.B. durch schlechten Programmierstil)
- Projektmanagement

Fazit – Bedeutung des SW-Engeneerings

Bedeutung des SW-Engineerings bei der Entwicklung

- Häufig wettbewerbsdifferenzierende Merkmale
- Interdisziplinarität ⇨ enge Kopplung mit den anderen Disziplinen
- Qualität der SW spielt enorme Rolle
- SW-Entwicklung birgt hohe Kosten

⇨ **SW-Engineering bei Entwicklung komplexer Systeme unverzichtbar!**

Bedeutung des SW-Engeneerings

Unterschiede zu „Standard"-SW-Entwicklung, v.a. hinsichtlich:

- Echtzeitfähigkeit
- Begrenzt HW-Ressourcen
- Entwicklungs- und Zielplattform sind unterschiedlich
- Verteilte Systeme ⇨ Kommunikationsbedarf
- SW als Teil eines „Ganzen"⇨ Erhöhter Testaufwand

⇨ Ähnliche Vorgehensweise, wie bei „Standard"-SW-Entwicklung

Aber:
⇨ Andere Rahmenbedingungen (Performance, Ressourcen, etc.)

⇨ Wissen über den Gesamtprozess erforderlich!

Danke für Aufmerksamkeit!