Software-Engineering

Vorlesung 11 vom 10.01.2005

Sebastian Iwanowski
FH Wedel

Software-Engineering

Vorlesungsthemen:

1. Überblick über das Thema und die Vorlesung
2. Grundlegende Prinzipien
3. Softwareplanung
4. Systemanalyse
5. Softwareentwurf
6. CASE-Tools (UML und ARIS)
7. Aufwandsabschätzung
8. Qualitätsmanagement
9. Projektmanagement

Dynamische Verfahren zum Softwaretest

Grundsätzlicher Verfahrensunterschied:

Blackbox-Test

- Ableitung der Testfälle aus der Spezifikation des Systems
- Innere Struktur des Programms wird nicht beachtet.

Whitebox-Test

- Ableitung der Testfälle aus der Struktur des Programms
- Es wird angestrebt, möglichst alle Programmstrukturen zu durchlaufen (kontrollflussbezogenes Verfahren).
- Es wird angestrebt, möglichst alle Datenklassen zu testen, zwischen denen das Programm unterscheidet (datenflussbezogenes Verfahren).

Whitebox-Test

Kontrollflussbezogene Verfahren

Anweisungsüberdeckung:

- Alle Anweisungen werden mindestens einmal ausgeführt.

Zweigüberdeckung:

- Alle Verzeigungen im Kontrollfluss werden mindestens einmal verfolgt.

Bedingungsüberdeckung:

- Alle booleschen Wertekonstellationen von (Teil-) Bedingungen werden einmal berücksichtigt.

Pfadüberdeckung:

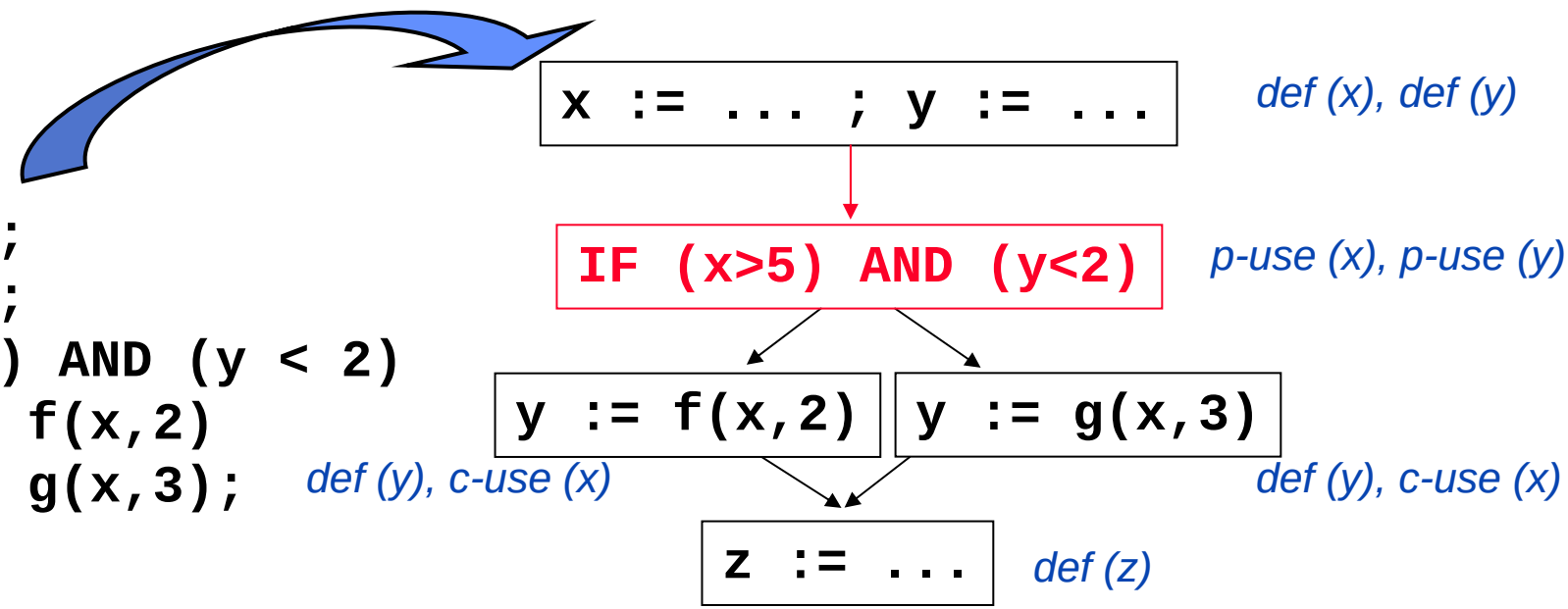
- Durchlaufen aller Pfade von Start- zum Endknoten:
  Das ist außer bei sehr kleinen Programmen nur theoretisch möglich.

Whitebox-Test

Datenflussbezogene Verfahren (Bsp.: defs/uses-Verfahren)

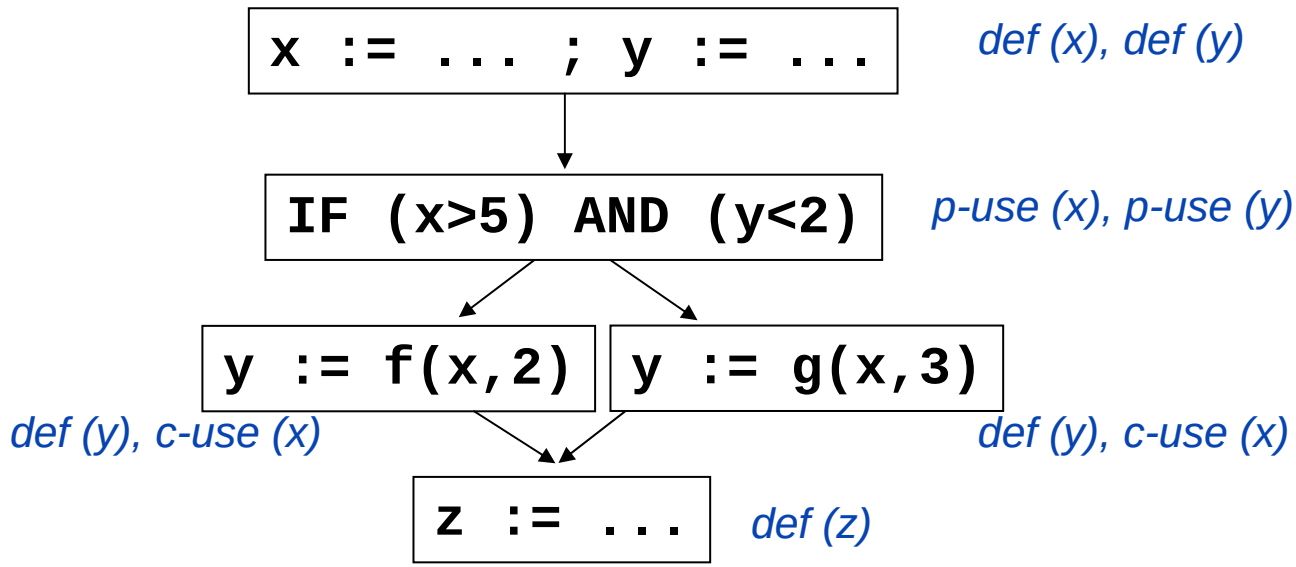
- Erweiterung der Kontrollflussgraphen um Datenflüsse
- Unterscheidung verschiedener Typen von Variablenzugriffen:
  - in Zuweisungen für die zugewiesene Variable (def)
  - in Wertausdrücken für ausgewertete Variable (c-use)
  - in Bedingungen für ausgewertete Variable (p-use)

```
x := ... ;
y := ... ;
IF (x > 5) AND (y < 2)
THEN y := f(x,2)
ELSE y := g(x,3);
z := ...
```



Whitebox-Test

Datenflussbezogene Verfahren (Bsp.: defs/uses-Verfahren)



- Betrachte „definitionsfreie Pfade“: Bis wohin wirkt sich die Zuweisung einer Variablen aus ?
- Unterscheidung verschiedener Typen von Datenflussüberdeckungen:
  - all defs: Jede Definition wird mindestens einmal benutzt
  - all p-uses: Jede Kombination Definition / p-use wird getestet
  - all c-use: Jede Kombination Definition / c-use wird getestet

Whiteboxtest von größeren Programmen

- Die vorgestellten Verfahren sind nur bei kleinen Programmen durchführbar.
- Größere Programme sind ohnehin modular aufgebaut

  => sie bestehen aus vielen kleinen Programmen.
- Teste die Module einzeln !
  (wird meistens bereits bei der Implementierung gemacht)
- Außerdem müssen die **Modulverbindungen** getestet werden:
  - Von übergeordneten Programmen aus:
    Simulation der untergeordneten Module durch so genannte **Stub-Module**
  - Von untergeordneten Modulen aus:
    Simulation der Einbettung durch so genannte **Treiberprogramme**

Blackbox-Teststrategien

Domain-Tests	"domain driven"
Stress-Tests	"stress driven"
Tests gegen Spezifikation	"specification driven"
Tests mit Zufallsdaten	"random / statistical tests"
funktionale Tests	"function driven"
Anwender-Tests	"user testing"
...	
...	
...	

Domain-Tests

Schlagworte

- Wertebereiche
- Äquivalenzklassenbildung allgemein

Vorgangsweise

- Testdaten aus häufig auftretenden Ereignissen auswählen
- Repräsentative Werte aus großen Wertebereichen festlegen

Musterbeispiele

- Äquivalenzanalyse von einfachen numerischen Eingabefeldern
  (Bsp. für Eingabebereich: *Zahlenbeträge, Punkt-Komma-Regelungen*)
- Drucker-Kompatibilitätstests
  (Bsp: Kompatibilität für alle HP-Drucker)

Domain-Tests

Stärken	Schwächen
• Die am wahrscheinlichsten auftretenden Fehler können mit einem relativen kleinen Testaufkommen erreicht werden. • Klares intuitives Vorgehen, generelle Verwendbarkeit	• „Blinde Flecken“ • Fehler, die sich nicht an Wertegrenzen befinden oder die sich in speziellen Optimierungszweigen befinden. • Oft sind die tatsächlichen Wertebereiche nicht bekannt.

Stress-Tests

Schlagworte

- Versuch, die Software zu überlasten

Vorgangsweise

- Das Produkt wird in den atypischen Fehlerfall oder in die Überlastung getrieben.
- Ziehen von Rückschlüssen auf das Programmverhalten unter normalen Umständen.

Musterbeispiele

- Hohes Datenaufkommen, LAN / WAN-Verbindungsabbruch, lange Transaktionsketten
- Geringe Speicherreserven, Rechnerausfall, Viren-Resistenz

Stress-Tests

Stärken	Schwächen
• Die Schwächen, die normalerweise erst beim länger dauernden Echteinsatz auftreten, werden rechtzeitig erkannt:	• „Blinde Flecken“ • Viele Fehler treten unabhängig von der Belastung auf. Der Stress-Test macht sie daher nicht „sichtbarer“.

Test gegen Spezifikation

Schlagworte

- „*Überprüfen wir, ob alles geht, was wir versprochen haben ...*“

Vorgangsweise

- Das Produkt wird auf jede Spezifikations- oder Designaussage überprüft.
- Das Pflichtenheft wird wieder durchgesehen.

Musterbeispiele

- Abdeckungsmatrix: stellt den Zusammenhang zwischen Testfällen und Spezifikationsdetails dar.
- Das Pflichtenheft kann auch die Testabnahmeprotokolle spezifizieren. (Beispiele: Serverbelastungsfähigkeit, garantierte Response-Zeiten)

Test gegen Spezifikation

Stärken	Schwächen
• Beste Absicherung gegen Gewährleistungsansprüche • Erhalt der Glaubwürdigkeit beim Kunden selbst bei Fehlerauftritt	• Bei Design- und damit verbundenen Spezifikationsschwächen überlebt das Produkt den Test genau dort, wo das Produkt seine Fehler hat.

Tests mit Zufallsdaten

Schlagworte

- Massentests mit immer neuen Testdaten

Vorgangsweise

- Testrechner erzeugt, führt aus und evaluiert eine große Anzahl von Tests.
- Es werden automatisch Zusammenhänge verschiedener Einzelfälle ermittelt (Regressionsanalyse).

Musterbeispiele

- Tests von Funktionen oder Subsystemen auf der Basis von vorhersagbaren Resultaten.
- Testaussage aufgrund statistischer Berechnungen bzgl. des technischen Abdeckungsgrads

Tests mit Zufallsdaten

Stärken

- unabhängig von immer wieder verwendeten Testdaten
- liefert bei neuen Programmteilen schneller Ergebnisse
- Übersehen von Sonderfällen weniger wahrscheinlich
- findet auch Fehler, die infolge langer Bearbeitungsketten entstehen

Schwächen

- Schwierigkeit bei der Diskriminierung zwischen „ok“ und „not ok“

  `(not crash = not fail) = false`

- Da diese Tests viele verschiedene Fehlerarten finden und dazu noch automatisch eine Regression bereitstellen, wird die Notwendigkeit von nichtautomatischen Tests oft übersehen.

Funktionale Tests

Schlagworte

- „Black Box Unit Tests“

Vorgangsweise

- Jede Einzelfunktion des Programms wird analysiert und getestet.

Musterbeispiele

- Tests von Spreadsheets (Test jeder Einzelfunktion)
- Tests von Datenbanken (Test jedes Berichts, jeder Abfrage, etc.)

Funktionale Tests

Stärken	Schwächen
• Genaue Betrachtung jeder Software-Einheit	• Zusammenspiel der Einheiten wird nicht getestet. Daher für Systemtests ungeeignet
• Einsatzbereich bereits vor dem Integrations- und Systemtest, daher werden die Fehler früher gefunden.	• Funktionstests werden häufig von den Entwicklern selbst geplant. Diese haben nicht die wirkliche „Blackbox-Mentalität“.

Anwender-Tests

Schlagworte

- „realitätsbezogen testen“

Vorgangsweise

- Anwender testet das gesamte System so, als ob es das fertige Endprodukt wäre.

Musterbeispiele

- „Beta-Tests“
- Unspezifierte manuelle Szenario-Tests vor Einführung der automatischen Tests mit Zufallsdaten.

Anwender-Tests

Stärken	Schwächen
• Design-Fehler werden sehr schnell sichtbar.	• Testabdeckung ist sehr ungewiss.
• Diejenigen Schwachstellen des Programms werden aufgezeigt, die zu Unverständnis führen oder hohe Fehleranfälligkeit seitens des Benutzers nach sich ziehen.	• Die Qualität der Testfälle ist ungesichert. Die Testfälle könnten zu trivial sein, um subtile Fehler finden zu können.
• Inhouse-Beta-Tests erlauben gezielte Nutzungsstudien und geben Hinweise auf weitere Anwenderbedürfnisse.	• Beta-Tests kosten Geld: entweder über Vertrieb und Imagekosten oder über die Bezahlung einer eigenen Inhouse-Beta-Abteilung.

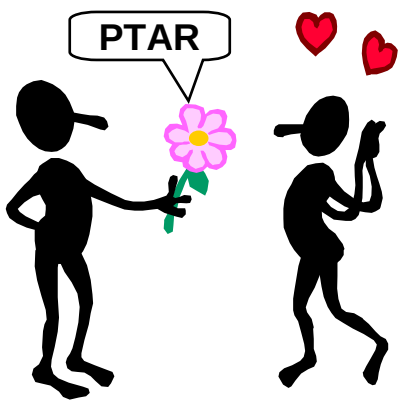
PTAR: Problem Tracking and Reporting

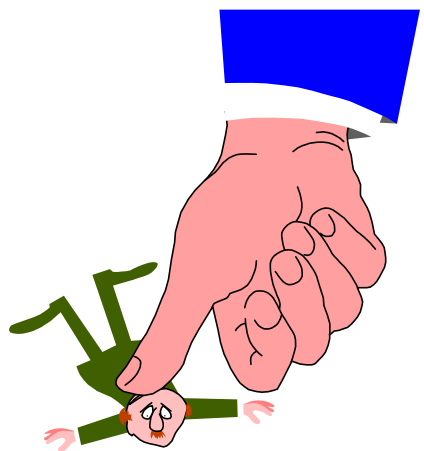
Standardisierung eines dokumentierten Problems

wie überall, wo Menschen miteinander arbeiten:

Motivation durch gegenseitige Anerkennung

„Ich habe dir einen PTAR geschickt“ vs. „Du hast einen Fehler gemacht“



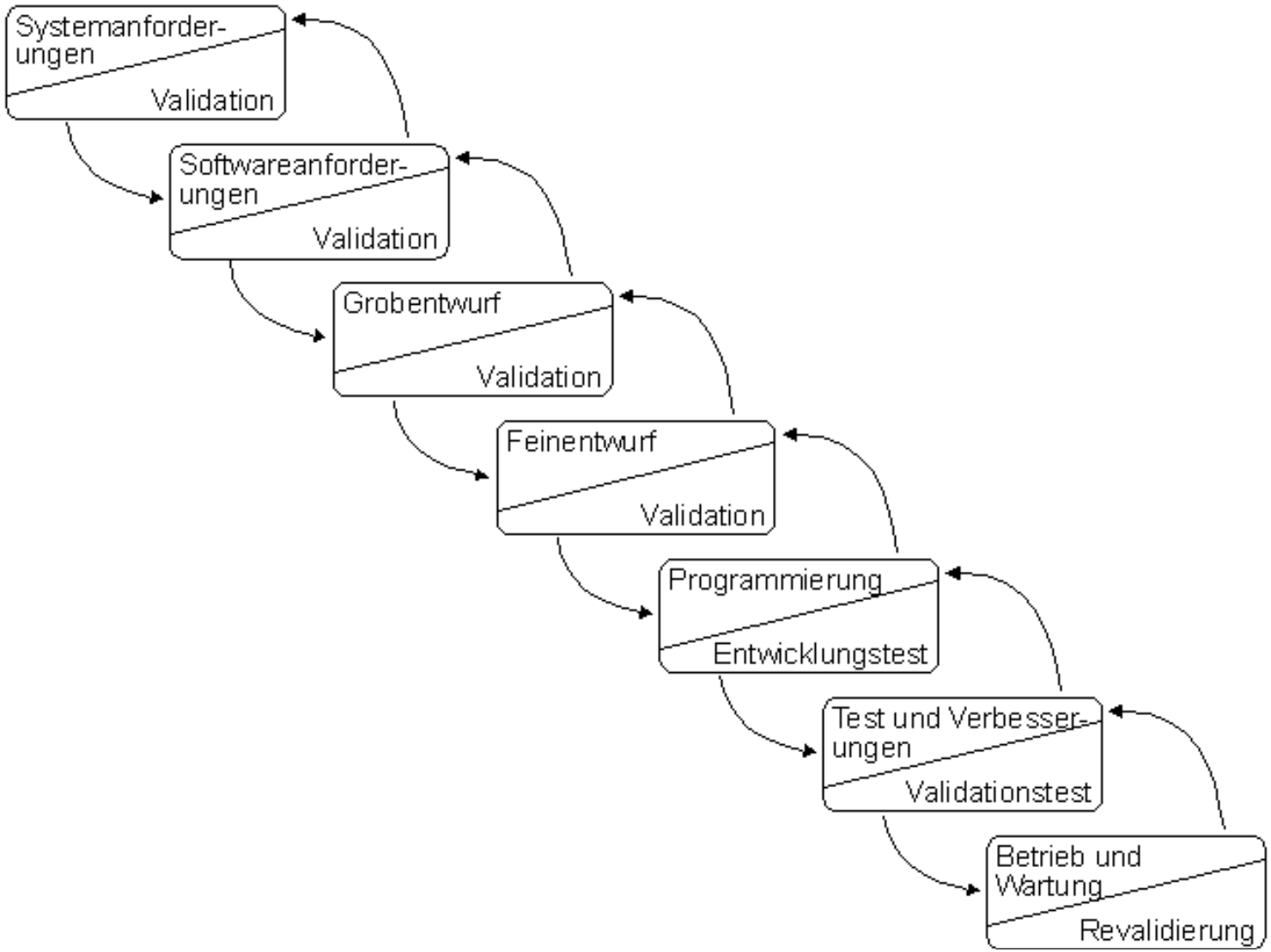


Software-Engineering

Vorlesungsthemen:

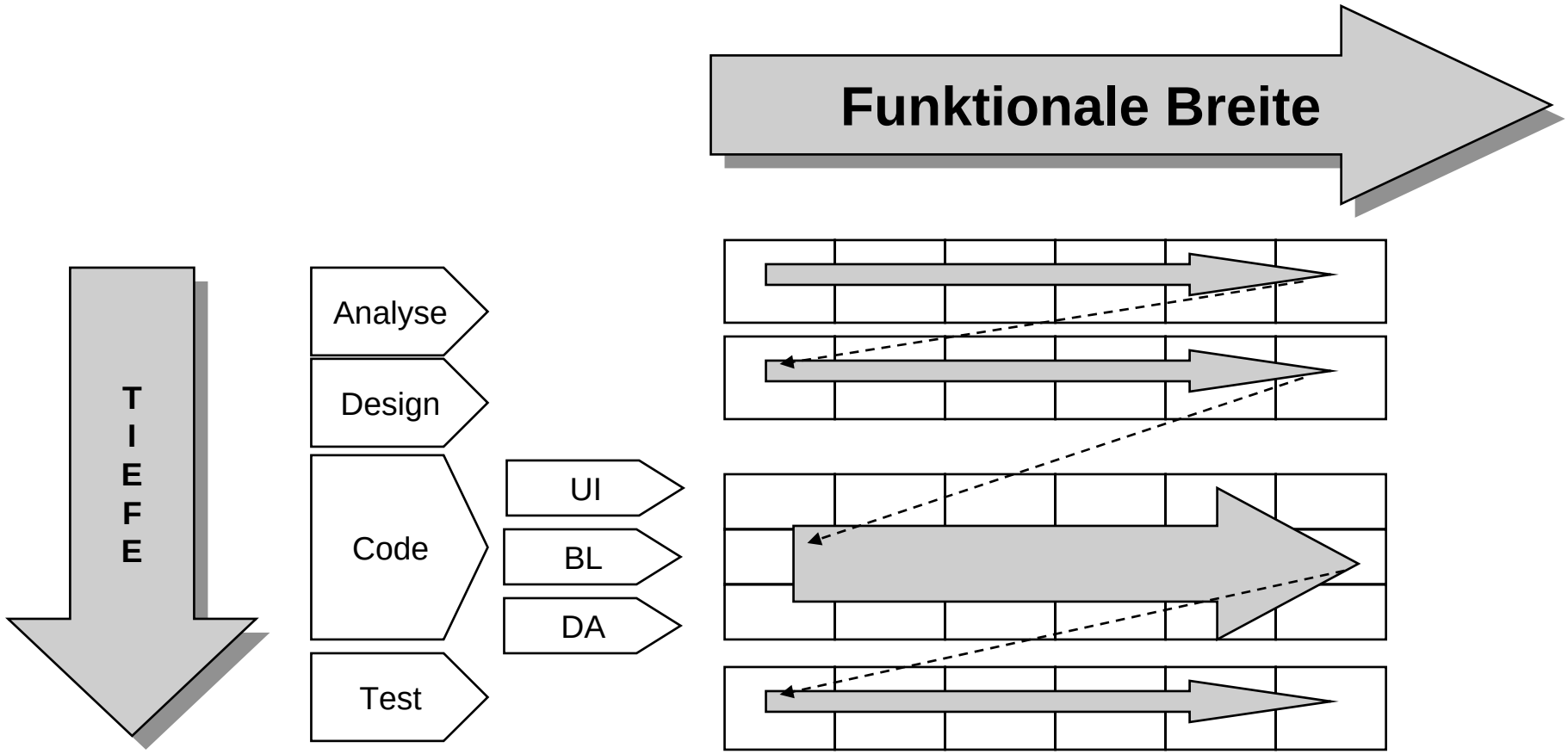
1. Überblick über das Thema und die Vorlesung
2. Grundlegende Prinzipien
3. Softwareplanung
4. Systemanalyse
5. Softwareentwurf
6. CASE-Tools (UML und ARIS)
7. Aufwandsabschätzung
8. Qualitätsmanagement
9. ➔ Projektmanagement

Wasserfallmodell



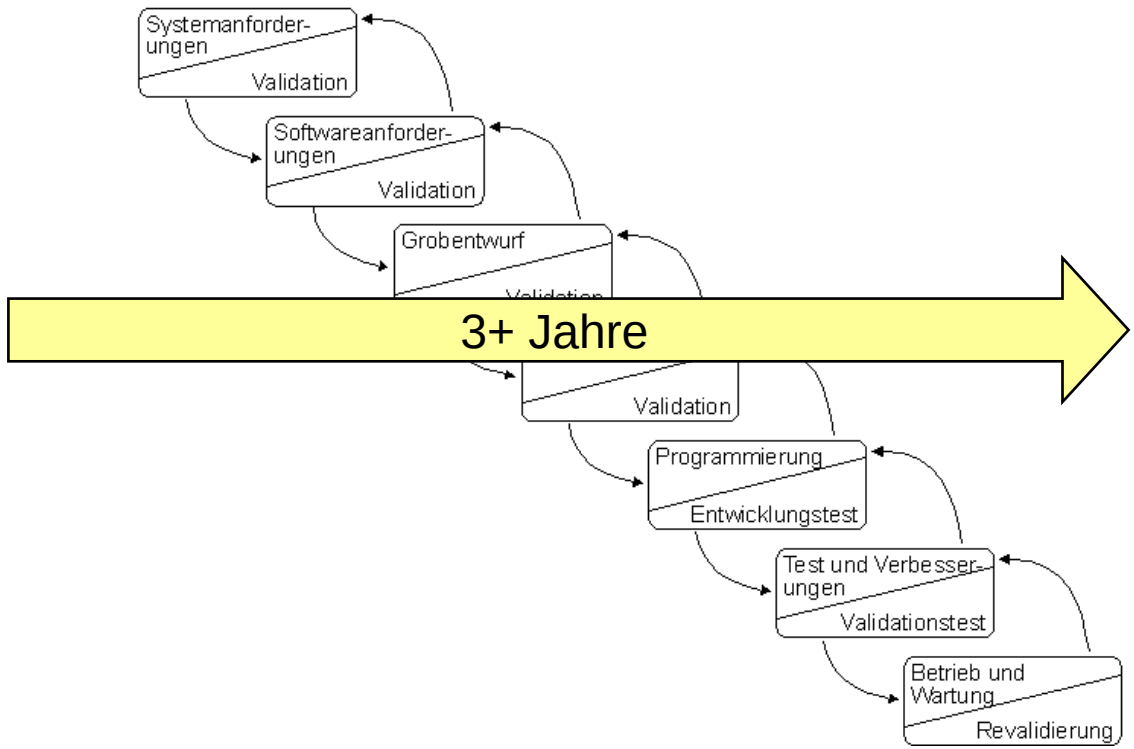
Wasserfallmodell

Zeitliche Einordnung in Breiten-/Tiefenraster



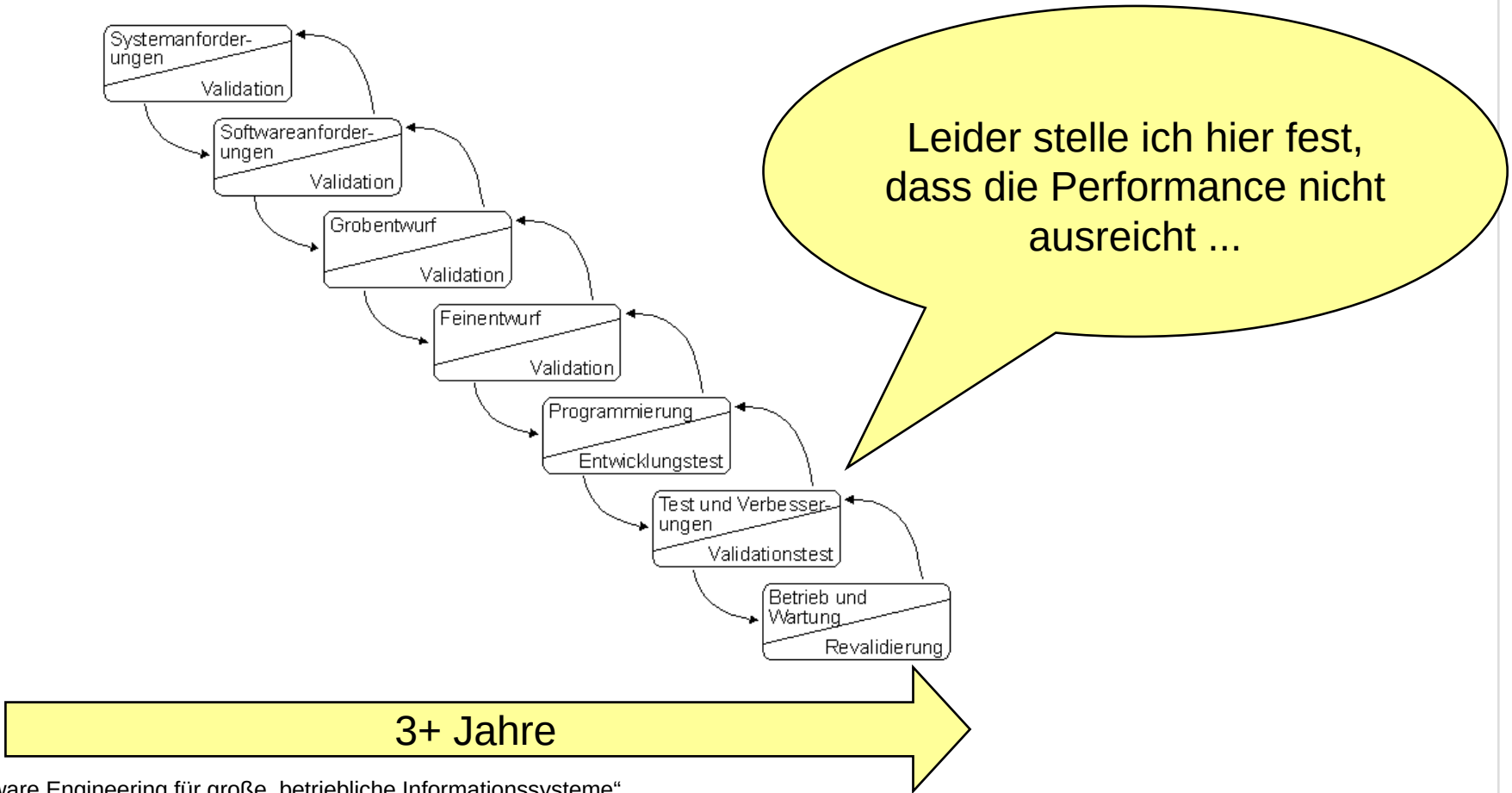
Wasserfallmodell

… ist für langfristige Produktentwicklungen gedacht



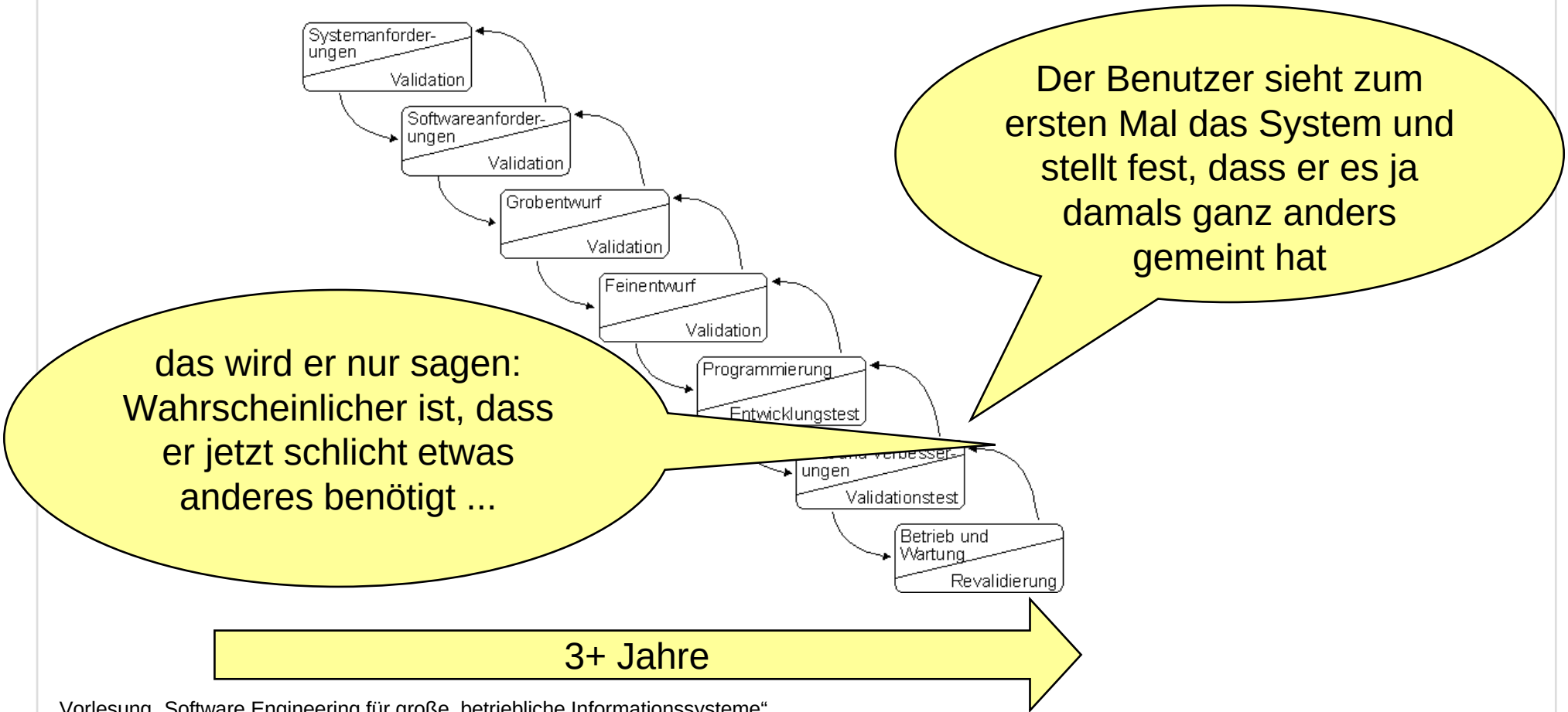
Wasserfallmodell

Langfristig heißt: Mit späten Konsequenzen …



Wasserfallmodell

Für die Softwareentwicklung: Mit zu späten Konsequenzen …



Wasserfallmodell

Vorteile

- Die Tätigkeiten der Systementwicklung werden in einer sinnvollen Reihenfolge angeordnet.
- Die Tätigkeiten und Dokumente können standardisiert werden. Damit können große Teams arbeiten. Das ganze ist einfacher zu managen.
- Phasenübergang erfolgt nach Validierung und Freigabe (Qualitätssicherung)
- Damit hat man Meilensteine und eine Fortschrittskontrolle – einfach zu managen.

Wasserfallmodell

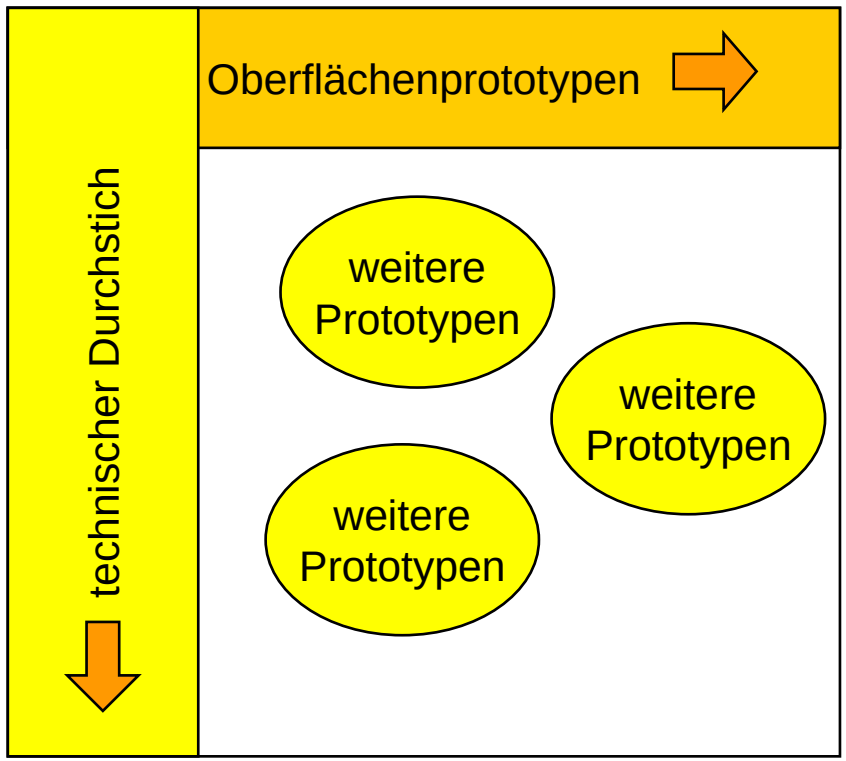
Nachteile

- Der Kunde sieht das Produkt sehr spät – bis dahin muss er es sich vorstellen – das überfordert 99%.
- Die meisten Kunden wollen keine Spezifikationen lesen – sie wollen Software.
- Linearer Durchlauf der Phasen ohne Änderungen ist sehr sehr unwahrscheinlich, weil sich wahrscheinlich die Vertragsgrundlage ändern wird.
- Zusammen mit „alten“ Programmierumgebungen hat man das Problem der exponentiellen Änderungskosten.
- Die Dokumente der einzelnen Phasen sind redundant. Änderungen der Geschäftsgrundlage werden über die Zeit nicht nachgezogen. Damit wird die Dokumentation früherer Phasen wertlos.

Prototyping

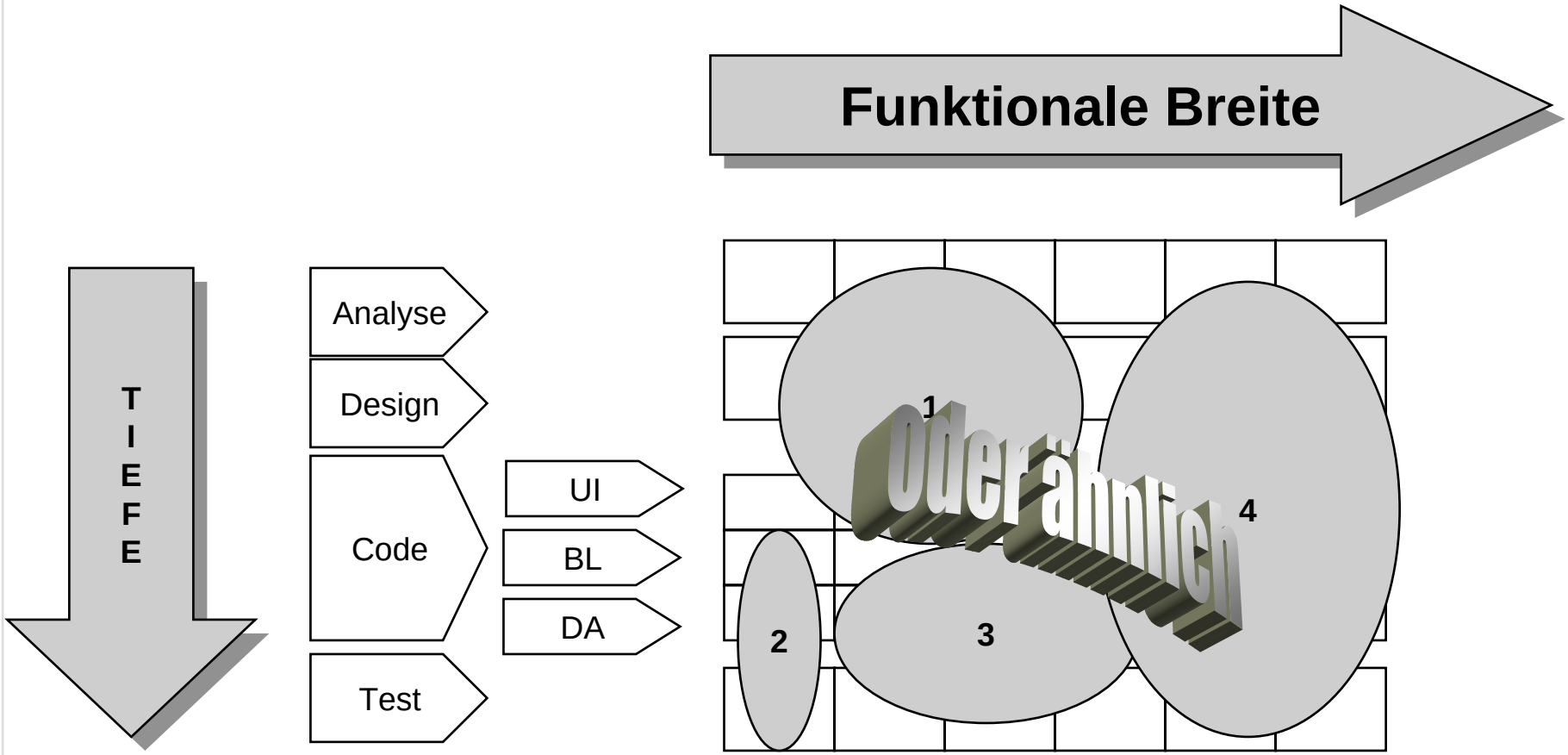
Fokussierung auf verschiedene Aspekte

- Durch technische (vertikale) Prototypen kann man vermeiden, dass man zum Beispiel die Performance-Überraschung zu spät erlebt.
- Durch horizontale Prototypen sieht der Kunde früher, was er bekommt.
- Komplexe Stellen sollte man mit weiteren Prototypen früh angehen („hardest things first“) – man merkt dann früh, welche Probleme man angehen muss.



Prototyping

Zeitliche Einordnung in Breiten-/Tiefenraster



Prototyping

Vorteile

- Man kann damit Risiken minimieren und frühzeitig Feedback vom Endbenutzer bekommen.
- Prototypen lassen sich gut in Prozessmodelle integrieren.
- Oberflächen-Prototypen lassen sich mit SW-Werkzeugen schnell bauen.
- Sie liefern wichtige Information über das Gesamtsystem (auch Grundlagen für FP-Schätzungen).

Prototyping

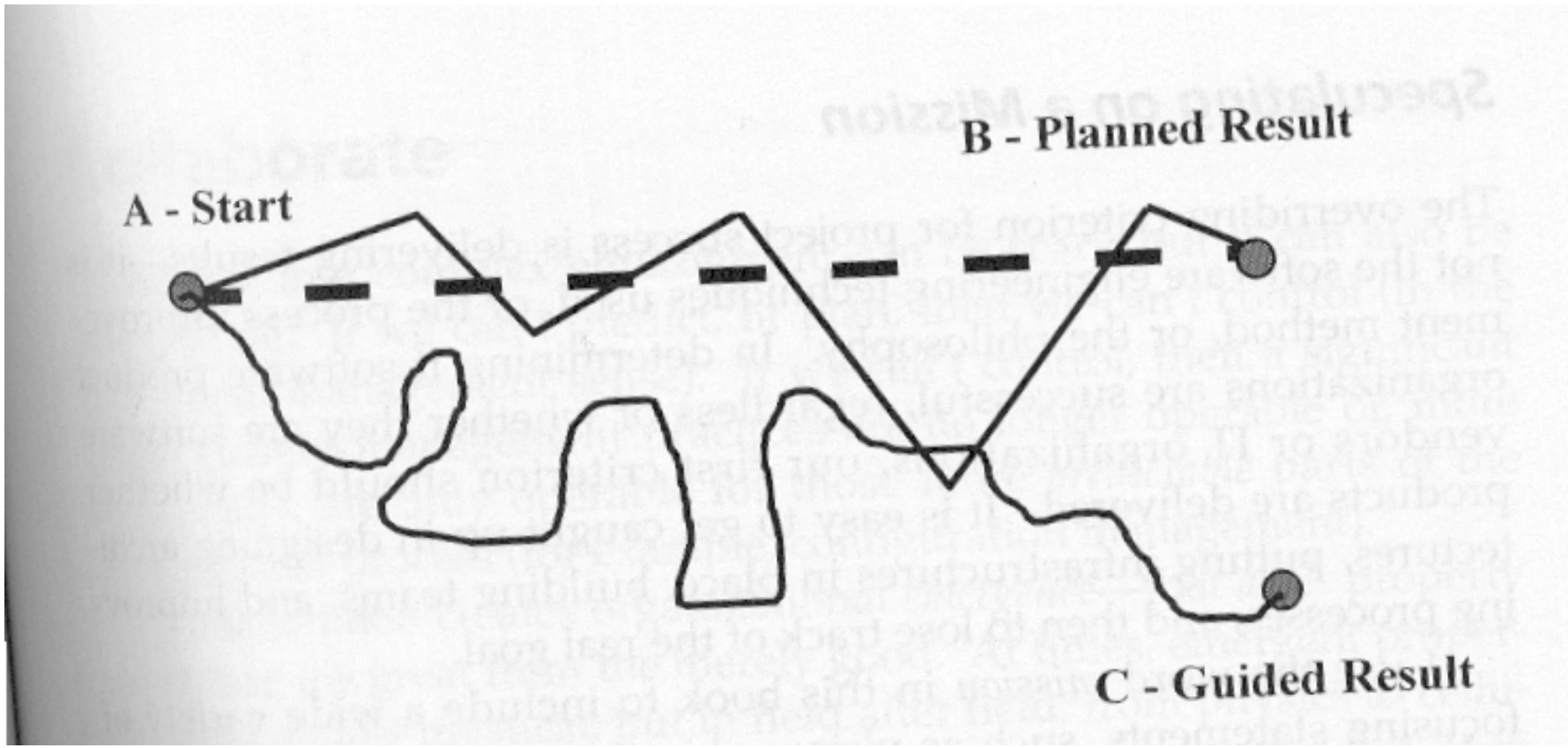
Nachteile

- Der Entwicklungsaufwand steigt, da von den Prototypen viel weggeworfen wird.

- Es soll schon vorgekommen sein, dass das Management einen Wegwerfprototypen als das fertige Produkt gesehen hat.
- und damit läuft man Gefahr, beim Management unter Druck zu geraten ...

Spiralmodell

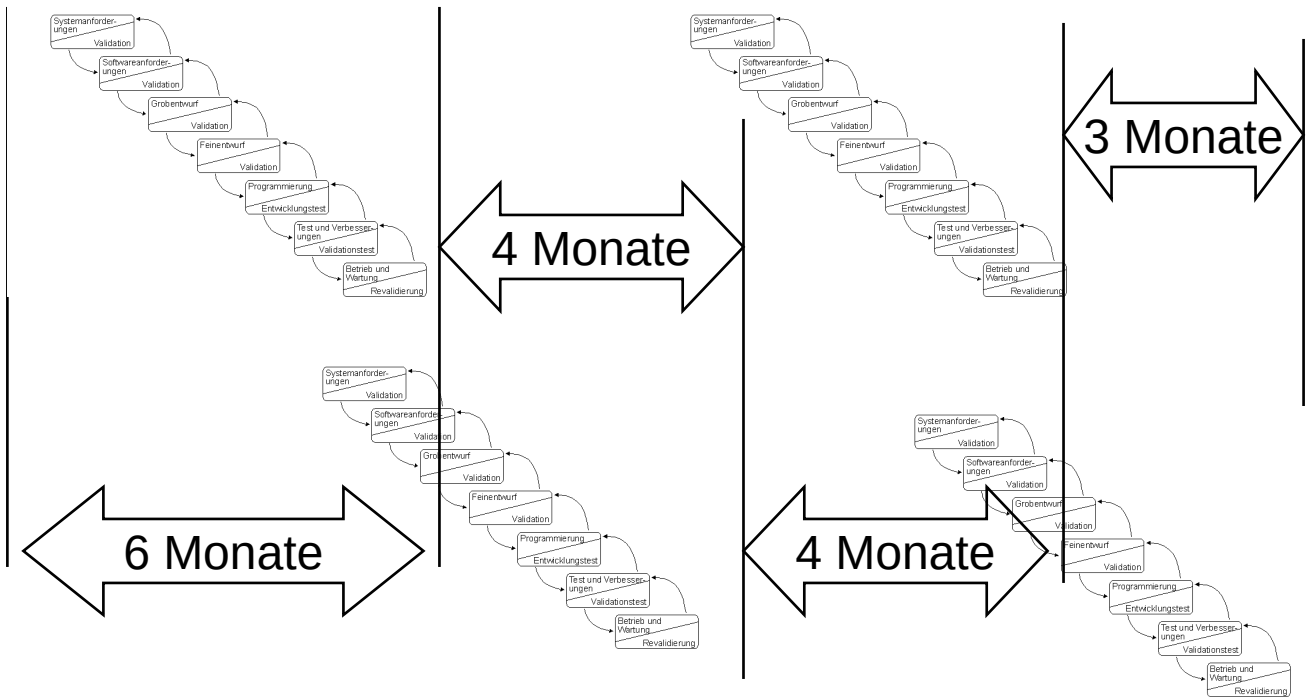
Wesentliches Problem des Wasserfallmodells: Moving Targets



[High2000]

Spiralmodell

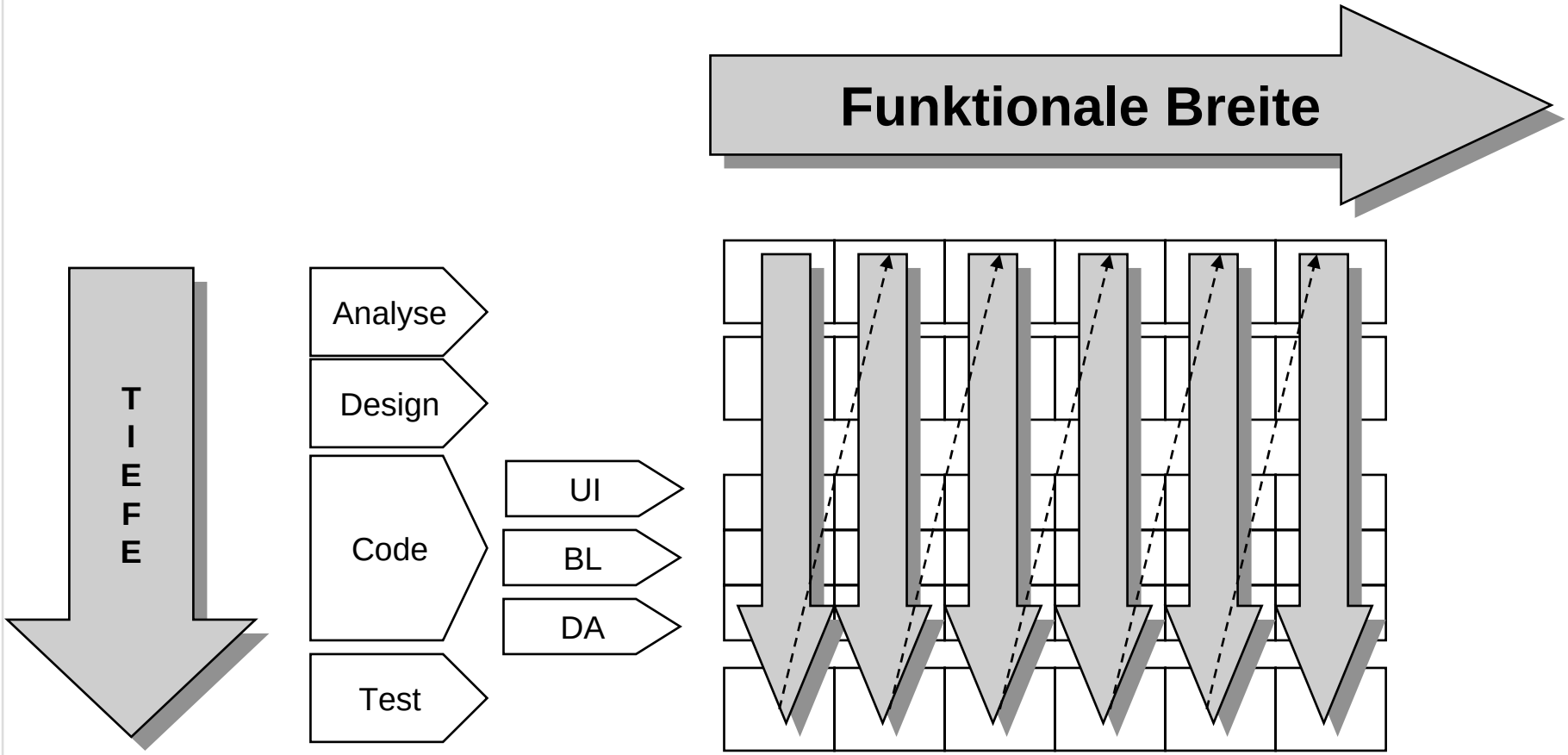
Idee: Aufteilen der Entwicklung in viele kurze Teilaufgaben



Überlappung
ist kein Zufall!

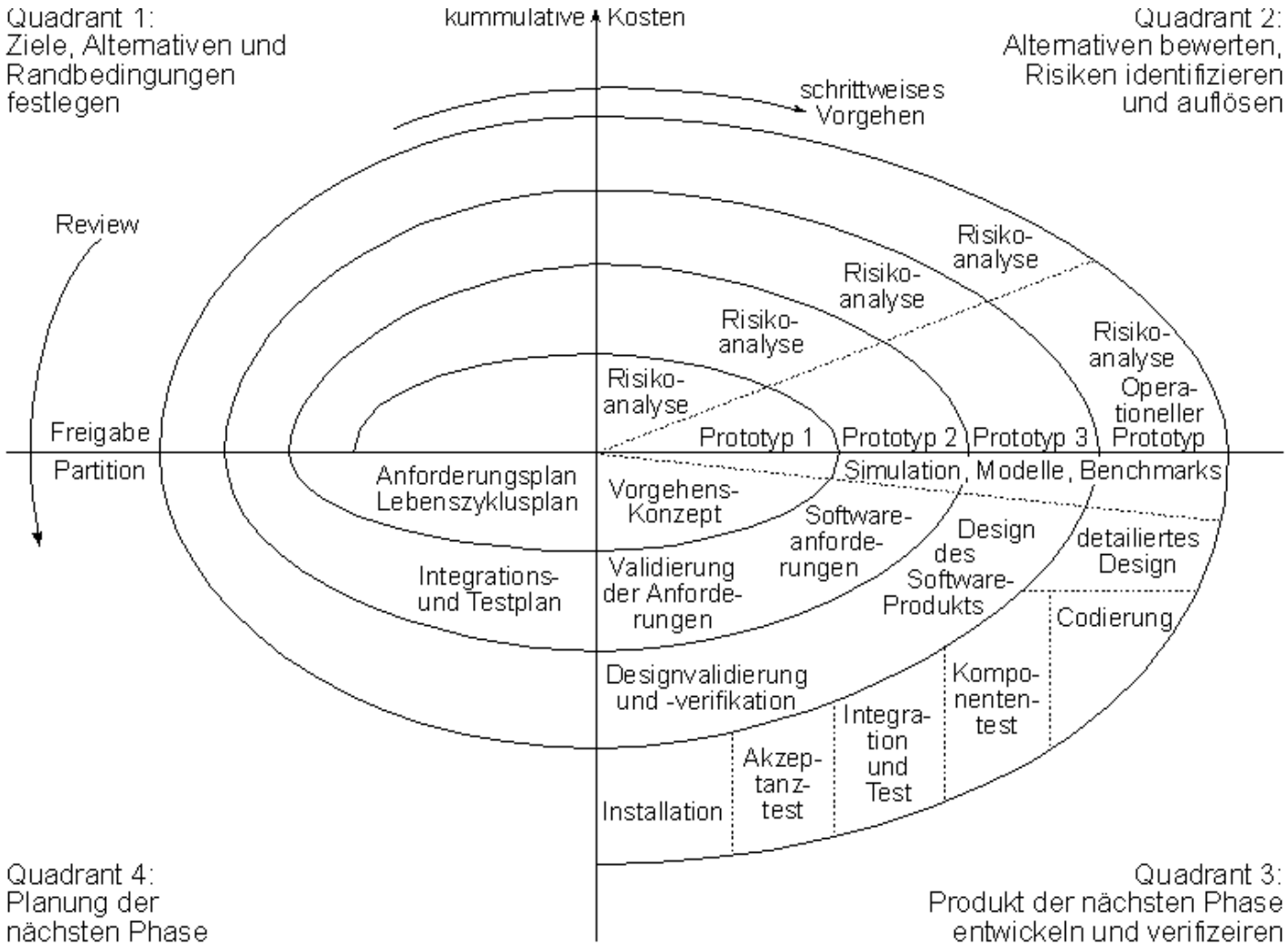
Spiralmodell

Zeitliche Einordnung in Breiten-/Tiefenraster



Spiralmodell

und so sieht es dann im Detail aus:



Spiralmodell

Vorteile

- Man bekommt permanentes Feedback – auf Moving Targets kann reagiert werden.
- Man kann pro Zyklus, wenn man möchte, ein anderes Prozess- und Teammodell wählen.
- Fehler werden relativ schnell erkannt.
- Man hat bessere Eingriffsmöglichkeiten als beim Wasserfallmodell.

Spiralmodell

Nachteile

- Man braucht für das Spiralmodell ein besseres Management
- Für kleine Projekte ist der Management-Aufwand unverhältnismäßig.

Beim nächsten Mal:
Projektmanagement (2. Teil)
Zusammenfassung der Vorlesung (für die Klausur)