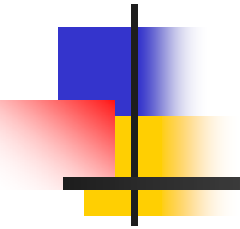


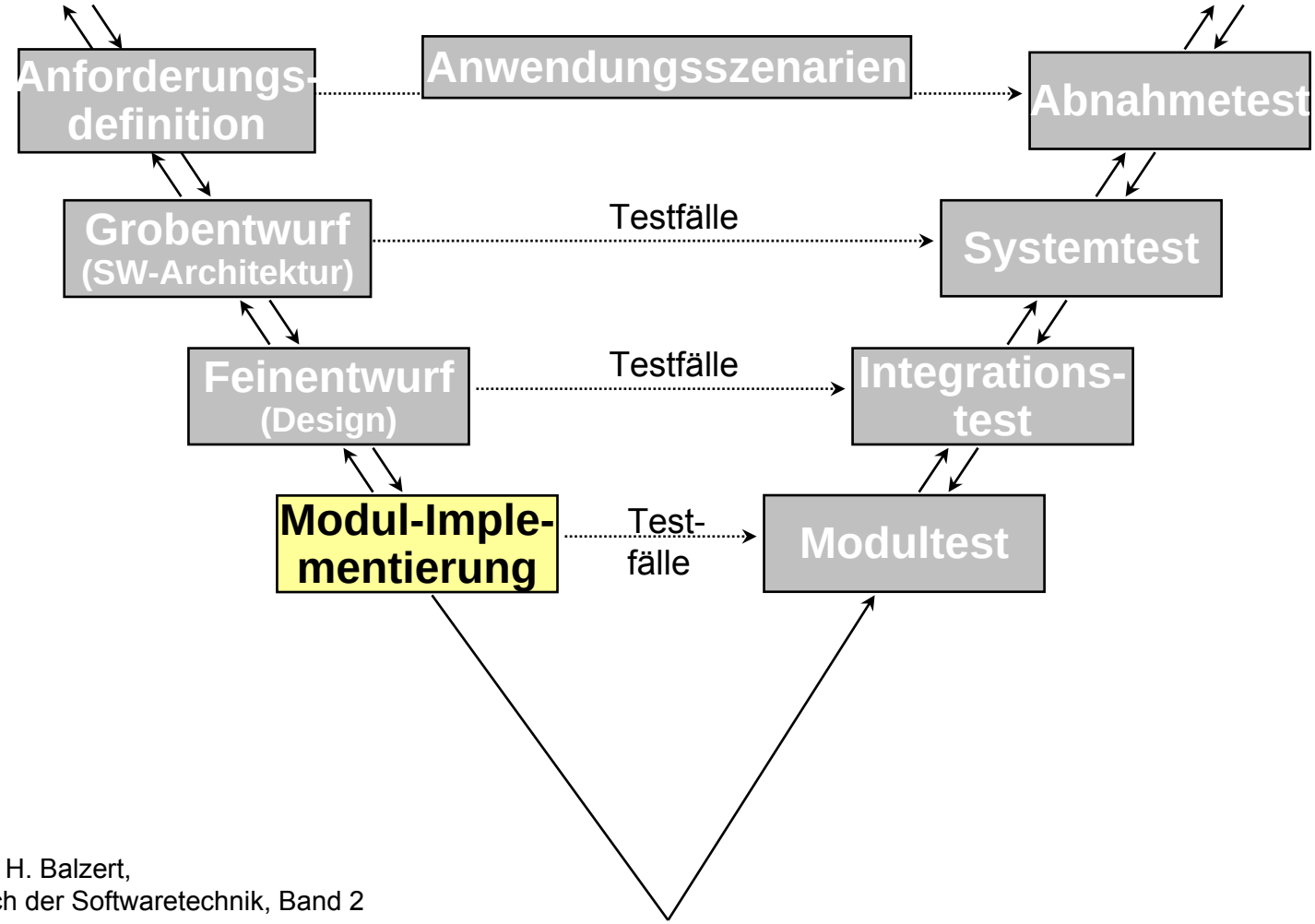
13. Implementierung

von

Prof. Dr. Wolfgang Weber

Folien teilweise übernommen von R. Hahn

Einordnung im V-Modell



Quelle: H. Balzert,
Lehrbuch der Softwaretechnik, Band 2

Lernziel Implementierung

Sie sollen in diesen Kapitel verstehen,

- **wo der Unterschied zwischen Implementierung in "Programmieren I & II"**
  und Implementierung in "SW-Technik I & II" liegt
- **warum Programmierrichtlinien wichtig sind**
- **was Programmierrichtlinien festlegen**
- **warum Programmierrichtlinien unbeliebt sind**

Anschließend können Sie in einem Team an einem großen Projekt mitarbeiten!

Implementierung

- In Programmieren I und II haben Sie gelernt
  - Konzeption von Datenstrukturen und Algorithmen
  - Strukturierung von Programmen
  - Dokumentation der Lösung und der Implementierung
  - Umsetzung der Konzepte in C++
  - Testen der entwickelten Software
  - allerdings für kleine Projekte in kleinen Teams ("Programmieren im Kleinen")
- In SW-Technik geht es um "Programmieren im Großen"
  - die erlernten Implementierungstechniken bleiben gültig
  - aber die Randbedingungen ändern sich!
  - Wartbarkeit und Beherrschung der Komplexität dominieren die Entwicklung!

Damit ein großes Projekt entwickelt werden kann, müssen gemeinsame Spielregeln vereinbart werden!

Bedeutung von Programmierrichtlinien

- Ziele
  - Zusammenarbeit in einem Projekt erleichtern
  - Integration und Test erleichtern
  - Langjährige Wartung ermöglichen
  - Zuverlässigkeit und Robustheit erhöhen
  - Steigerung der Effizienz der Entwicklung

- Richtlinien
  - sind verbindlich für den Entwickler
  - sollten nach Möglichkeit eingehalten werden
  - werden bei Beauftragungen zur Auflage gemacht

- die Einhaltung der Richtlinien wird überprüft
- Abweichungen sind nur mit (guter) Begründung erlaubt

Exemplarischer Inhalt einer Programmierrichtlinie (mit Beispielen)

- Allgemeines
  - Es ist nach dem ISO C++-Standard zu programmieren
- Kommentierung
  - Jede Klasse muss am Ort ihrer Deklaration mit einem Kommentar eingeleitet werden, der Name und Zweck der Klasse, Autor, Erstellungsdatum und Version sowie eine Änderungsliste enthält
- Namenskonventionen
  - Die C++-Schlüsselwörter sind als Symbolnamen (auch in Großbuchstaben) nicht zulässig
- Deklarationen
  - Bei paarweise existierenden Operatoren wie *==* und *!=* muss, falls einer überladen wird, auch der andere überladen werden
- Fehlerbehandlung
  - Der Rückgabewert von *new* muss vor der ersten Verwendung geprüft werden
- Sonstiges
  - Statt der Funktionen alloc() und free() sind *new* und *delete* zu verwenden
  - Bei Bedarf sind die Standard-Datentypen *string*, *vector* und *set* zu verwenden

Inhalt einer Programmierrichtlinie: Weitere Themen

- Festlegung
  - der I/O-Methoden
  - der verwendeten Zeichensätze
  - von erlaubten Dateinamen (z.B. 8.3)
  - von Variablennamen mit Datentypen (z.B. uint32MyNumber)
- Regeln
  - zur Verwendung von Zeigern
  - zur Verwendung von Sichtbarkeiten / Zugriffsrechten
  - zum Umgang mit dem Preprozessor
  - zum Aufteilung des Codes auf Dateien
  - zum Umgang mit Compiler-Warnungen
  - zur Art der Fehlerbehandlung
  - zur Initialisierung von Variablen
  - zu Layout, Formatierung des Codes
- und vieles mehr – je nach Projekt und Bedarf!

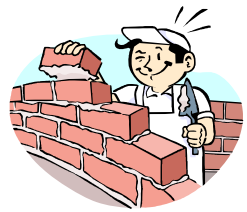
Vorteile einer Programmierrichtlinie

- Wartbarkeit
  - erhöht die Verständlichkeit des Codes für Teammitglieder und Neulinge
  - reduziert Bedarf an speziellem Fachwissen
- Integrierbarkeit
  - Vermeidung von spät auffallenden Inkompatibilitäten von Bibliotheken
  - Einheitliche Aufteilung des Codes auf Quellcode-Dateien
- Zuverlässigkeit
  - Regeln zum Umgang mit fehlerträchtigen Konstrukten (z.B. Pointern)
- Robustheit
  - Initialisierungsregeln, Zuweisungsregeln
- Testbarkeit
  - Einheitliche Fehlerbehandlung
- Effizienz der Entwicklung
  - Bietet eine Kommunikationsbasis
  - Empfehlung von anerkannten / bewährten Elementen
  - Vermeidung von bekannten fehleranfälligen Sprachkonstrukten
  - Arbeitsanleitung: konkrete Vorgaben z.B. für zu verwendende Datentypen
  - Erleichtert die Übernahme des Codes bei Entwicklerfluktuation

Nachteile einer Programmierrichtlinie

- Echte Nachteile:
  - Einschränkung der Individualität des Entwicklers
  - Gewöhnung an die vorgegebene Lösungen erforderlich

- beliebte Ausreden zur Umgehung:
  - "Schönes Programmieren ruiniert die Performanz"
    → der Unterschied wird von heutigen Compilern weg-optimiert!
  - "Kommentare in halbfertigem Code sind sinnlos"
    → grundlegende Entscheidungen sollen im Modell getroffen werden
  - "Die Anpassungen mache ich, wenn ich fertig bin"
    → in einem professionellen Projekt nicht akzeptabel



- Oft Widerstand – speziell von den "Programmier-Profis"

Die Richtlinien müssen als etwas Positives empfunden werden!

- Erstellung zusammen mit den Programmier-Profis
- unter Beteiligung aller Projektparteien
  (Entwickler, Tester, Integratoren, Qualitätssicherung...)

Kontrollfragen Implementierung

- **Warum sind Programmierrichtlinien oft unbeliebt?**
- **Wie kann man den Widerstand der Entwickler reduzieren?**
- **Warum wird die Einhaltung von Regeln höher gewichtet als Effizienz, Stimmung und Entwicklungsgeschwindigkeit?**
- **Was passiert in einem großen Projekt, wenn es keine Programmierrichtlinien gibt?**

Könnten Sie in einem Team an einem großen Projekt mitarbeiten?