Verbesserte Softwarequalität durch effiziente Testprozesse

Prof. Dr. Jan Peleska
TZI und Verified Systems International GmbH

Übersicht

- **Vorgehensmodelle:** Der Testprozeß im Software-Engineering
- **Testarten** auf unterschiedlichen Systemebenen: Unittest – Integrationstest – Subsystemtest – Systemtest
- **Test-Techniken:** statische und dynamische Tests, Testabdeckung, Metriken
- **Spezifikationsbasiertes Testen** am Beispiel von Business Applications und Embedded Systems
- **Tools** und Auswahlkriterien
- **Design** for Testability

Vorgehensmodelle: Der Testprozeß im Software-Engineering

Vorgehensmodelle (V-Modelle) ...

- ... regeln den Entwicklungsprozeß durch Definition von **Aktivitäten** und **Produkten**, die in den einzelnen Entwicklungsphasen durchzuführen bzw. herzustellen sind,
- ... geben Hinweise über die mögliche Verteilung der Verantwortlichkeiten bei der Durchführung der Aktivitäten,
- ... sind **generische** Regelungen, die für das spezifische Entwicklungsvorhaben instantiiert werden **(Tayloring)**.

Vorgehensmodelle mit Softwarebezug – Beispiele

- **DIN ISO 9000 Teil 3** schreibt **Testen und Validierung** als festen Bestandteil des Qualitätssicherungssystems vor.
- **RTCA DO 178B** (V-Modell für Softwareentwicklung in der zivilen Luftfahrt) unterscheidet zwischen den Prozessen **Verifikation** (aktive Durchführung von Verifikation, Validation und Test) und **Qualitätsmanagement** (Management aller Aktivitäten mit Qualitätsbezug).
- **V-Modell des Bundesinnenministeriums**[1] sieht Testen als Bestandteil des **Product Assessment (PA)**

[1] siehe z.B. `http://www.informatik.uni-bremen.de/~uniform/gdpa/`

Bestandteile des Product Assessment nach V-Modell

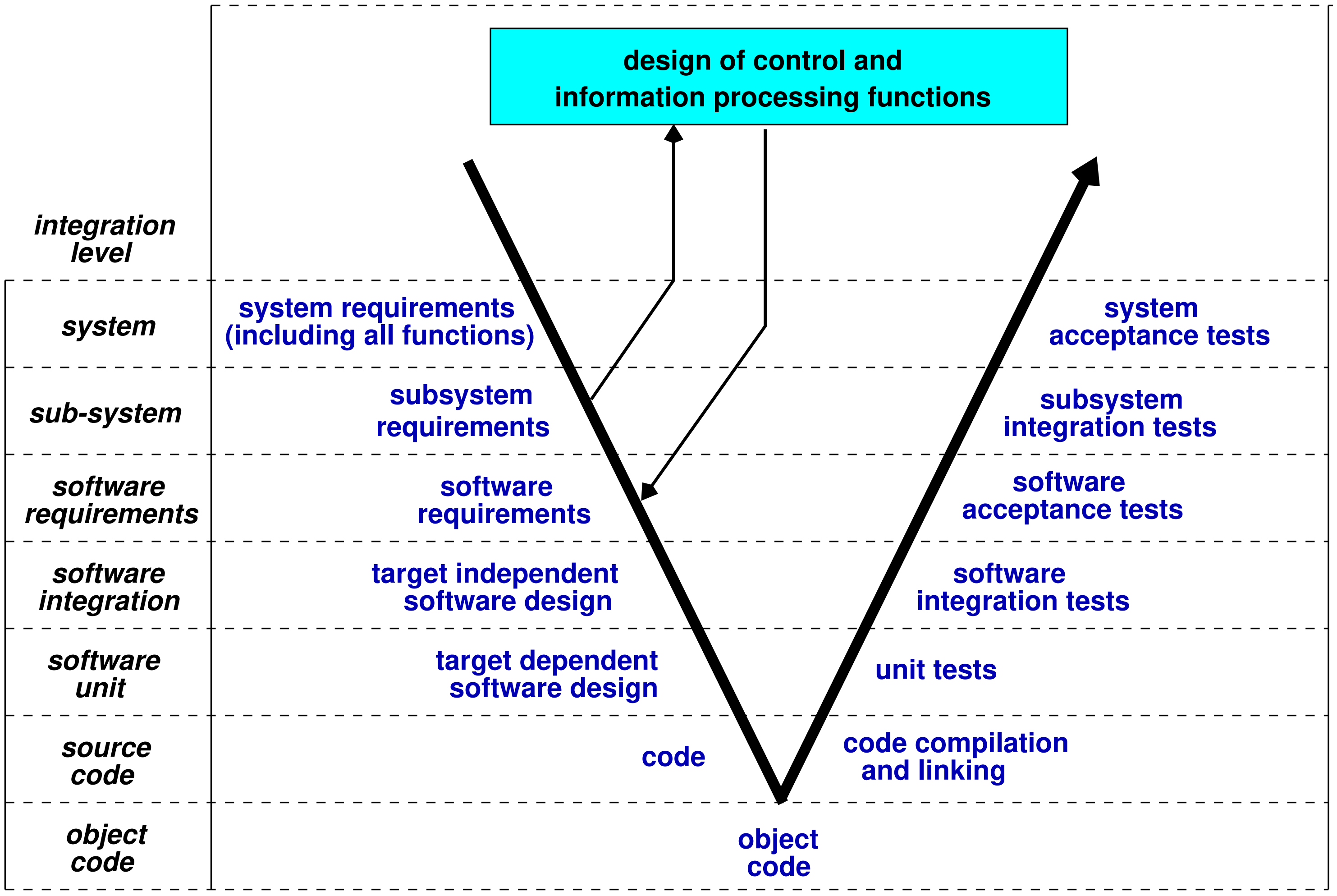
- Statische Analyse,
- Test,
- Simulation,
- Korrektheitsbeweis,
- Symbolische Programm-(Spezifikations-)ausführung,
- Review,
- Inspektion.

Grundforderungen des V-Modells zum Testen

- **Testfallbeschreibung:**
  - Was wird geprüft ?
  - Was sind die Randbedingungen/Anfangsbedingungen für den Test ?
  - Welche Eingabedaten sind für den Test erforderlich ?
  - Was sind die erwarteten Resultate ?

- **Überdeckungsmatrix:**
  - **Strukturüberdeckung:** Welche Architekturkomponenten werden durch den Test berührt ?
  - **Anforderungsüberdeckung:** Welche Nutzer-/Sicherheits-/technischen Anforderungen, werden getestet ?

Zuordnung: Testaktivitäten im Entwicklungsprozeß



Testarten – Gliederung

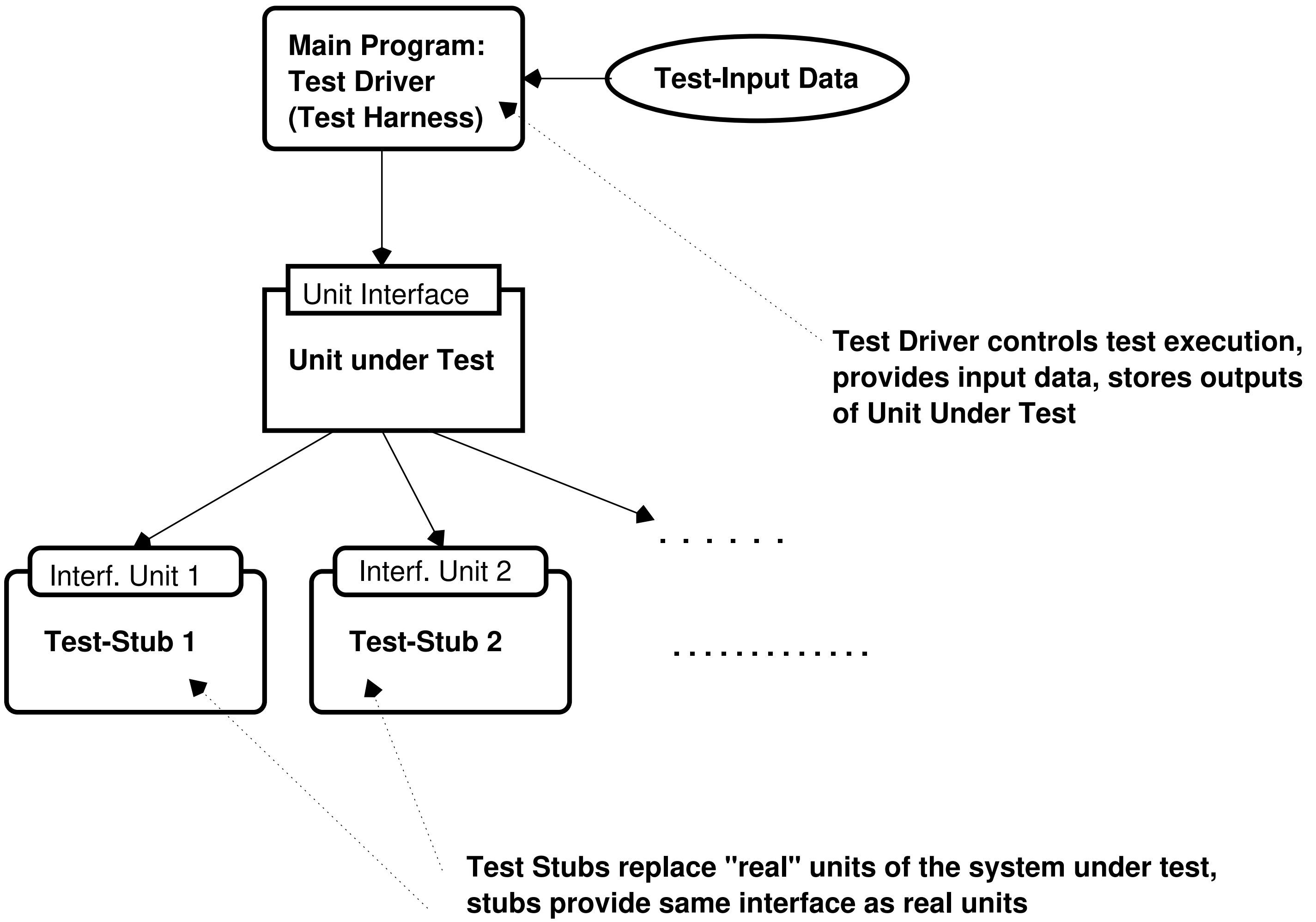
Die Testarten lassen sich nach folgenden Kriterien klassifizieren:

- **Integrationslevel:** Unittests – Integrationstests – Subsystemtests – Systemtests
- **Umgebungsbedingungen:** Test des Normalverhaltens – Test des Ausnahmeverhaltens
- **Korrektheitseigenschaften:** Funktionale Tests – Strukturtests
- **Beobachtungstiefe:** White-Box Tests – Black-Box Tests

Testarten – Unittests (= Modultests)

- **Testscope:** Unittests untersuchen die kleinsten “Bausteine” des Softwaresystems: Functions, Threads, Objects, Data Container …
- **Testziele:** Korrekte Arbeitsweise der Unit in Abwesenheit anderer zum System gehörigen Softwarekomponenten $\Longrightarrow$ die Unit wird isoliert geprüft.
- **Referenz:** Modulspezifikation, Feinentwurf

Testumgebung für Unittests



Testarten – Software-Integrationstest

- **Testscope:** Teilsysteme kooperierender Komponenten (Functions, Threads, Processes)
- **Testziele:** Korrekte Kooperation der Komponenten prüfen:
  - Datenfluß, Schnittstellen
  - Synchronisation beim Zugriff auf gemeinsame Ressourcen
- **Referenz:** Software-Subsystemanforderungen, Modulspezifikationen, SW-Architekturdesign (Grobentwurf)

Testarten – Subsystem-Integrationstest

- **Testscope:** HW/SW-Subsysteme (einzeln und kooperierend) – endet mit dem Test des Gesamtsystems
- **Testziele:** Korrekte Arbeitsweise der Subsysteme prüfen:
  - Zusammenspiel zwischen Hardware und Software im einzelnen Subsystem
  - Kooperation mehrerer Subsysteme
  - Funktionsweise der Kommunikationsmedien zwischen HW/SW-Subsystemen
- **Referenz:** Systemanforderungen, Subsystemanforderungen, Systemdesign

Testarten – Systemannahmetest

- **Testscope:** Gesamtsystem
- **Testziele:** Nachweis der spezifizierten Benutzeranforderungen $\Longrightarrow$ Systemannahmetest ist häufig eine Untermenge der auf Gesamtsystem-Level durchgeführten Integrationstests.

  Früher wurden Systemannahmetests auch mit dem Ziel der Validierung (d.h. Prüfung des Systems hinsichtlich seiner Wirksamkeit) durchgeführt. Eine so späte Validierung wird heute als viel zu riskant für den Projekterfolg angesehen.
- **Referenz:** Benutzeranforderungen

Testarten – Normalverhalten versus Ausnahmeverhalten

- **Normal Behaviour Tests** prüfen die Korrektheit des Testlings unter normalen Umgebungsbedingungen.
- **Exceptional Behaviour Tests (Robustness Tests)** prüfen die Korrektheit des Testlings in spezifizierten (d. h. vorhersehbaren) Ausnahmesituationen.
- Normal und Exceptional Behaviour Tests können auf jeder Integrationsstufen (Unit-Level bis System-Level) durchgeführt werden.

Testziele der Robustheitstests

Test-Level	**Testziele – Robust gegen ...**
Unit	falsche Eingabeparameter – falsche Datenobjekte
Integration	falsche Daten an Prozeß/Thread-Interfaces – falsche I/O-Folgen in der Kommunikation zwischen Komponenten – fehlende Ressourcen – Timeouts – Ausfall einzelner SW-Komponenten
Subsystem	fehlerhafte Kommunikationsprotokolle – lokale Hardwarefehler – lokale Überlast – Ausfall einzelner SW-Komponenten
System	Subsystemausfälle – Überlastsituationen im Gesamtsystem

Testarten – Black-Box versus White-Box

- **Black-Box Tests** untersuchen funktionale Eigenschaften der Komponente durch Analyse ihres **an der Schnittstelle sichbaren** Verhaltens.

  Black-Box Tests können i.a. nicht alle relevanten funktionalen Eigenschaften des Testlings prüfen ($\Longleftarrow$ nicht sichtbare interne Entscheidungen führen zu nicht-deterministischem Verhalten an der Schnittstelle)!

  – $\Longrightarrow$ erzielte Strukturüberdeckung beim Black-Box Test messen !

  – $\Longrightarrow$ ggf. zusätzliche White-Box Tests zum Erreichen der Strukturüberdeckung durchführen !

Testarten – Black-Box versus White-Box

- **White-Box Tests** untersuchen strukturelle Eigenschaften der Komponente.

  Für sinnvolle White-Box Tests müssen die Ergebnisse auch bzgl. ihrer funktionalen Korrektheit geprüft werden!

- Black-Box und White-Box Tests können auf jeder Integrationsstufen (Unit-Level bis System-Level) durchgeführt werden.

Test-Techniken

- Statische Tests – formale Verifikation
- Dynamische Tests
- Testabdeckung
- Metriken

Spezifikationsbasiertes Testen

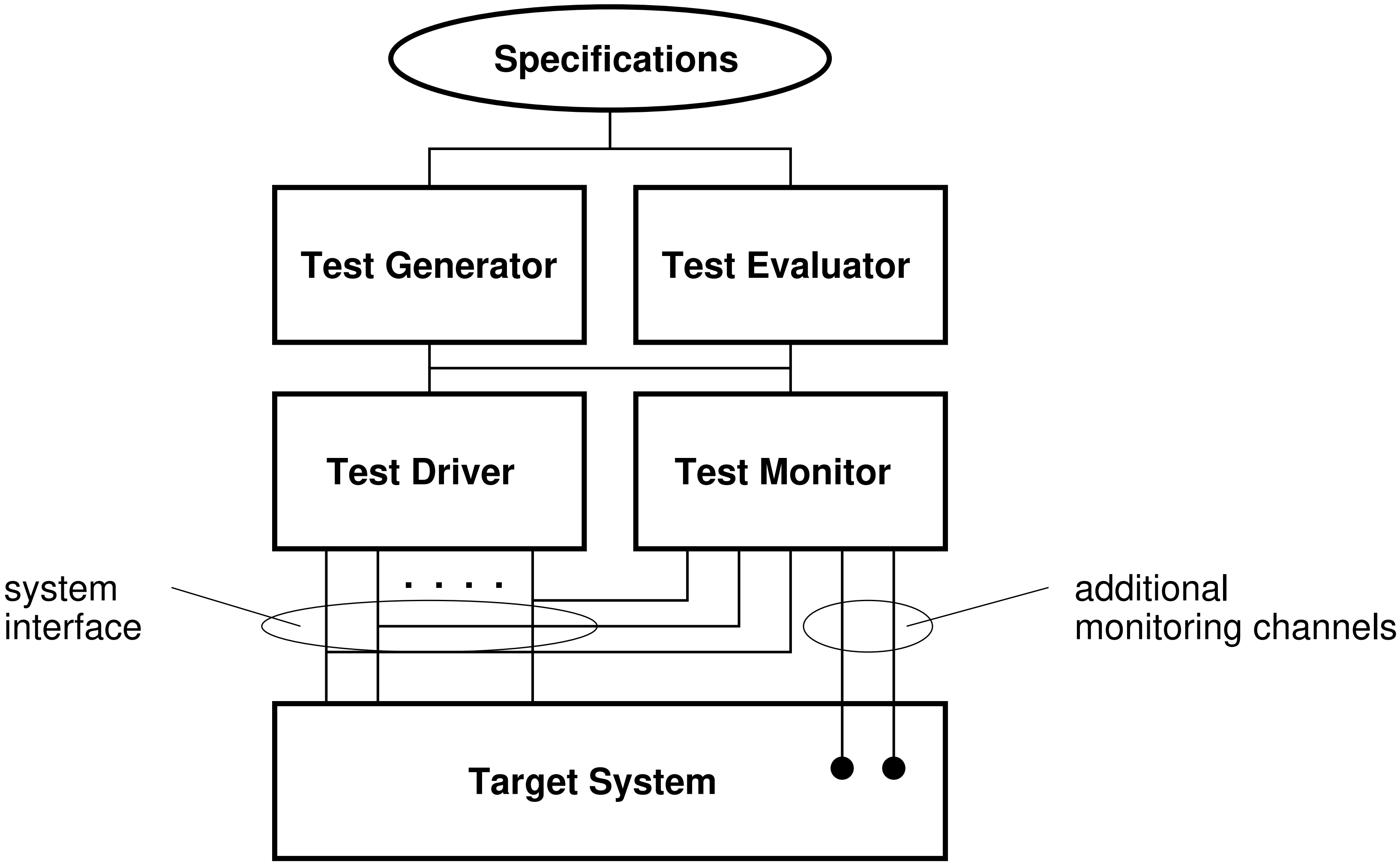
- **Spezifikationsbasiertes Testen auf Grundlage formaler – maschinell interpretierbarer – Spezifikationen ist die Voraussetzung für Automatisierung!**

Spezifikationsbasiertes Testen

Einige Fakten aus der Theorie:

- Anforderungen, die sich mit Hilfe paralleler, kooperierender Zustandsmaschinen formulieren lassen, erlauben automatische Testerzeugung, Durchführung und Auswertung in Echtzeit.
- Auch für nicht terminierende Systeme (z.B. Controller) läßt sich die Vollständigkeit der erzielten Überdeckung prüfen, wenn eine obere Schranke für die Anzahl der möglichen Systemzustände bekannt ist.
- Es gibt Anforderungen, die untestbar sind.
- Es gibt Anforderungen, die nicht automatisiert testbar sind.

Tools und Auswahlkriterien



Design for Testability – Kompositionalität

- **Kompositionalität:** Komponenten erfüllen ihre Spezifikation unabhängig davon, ob weitere Komponenten parallel aktiv sind oder nicht.
  - **Beispiel "nicht kompositionell":** zwei Prozesse greifen unsynchronisiert auf globale Variable zu
  - **Beispiel "kompositionell":** zwei Prozesse greifen mittels Monitor auf globale Variable zu

Design for Testability – Kompositionalität

Auswirkung der Kompositionalität auf das Testen:

- Funktionstests können mit den isolierten Komponenten ausgeführt werden.
- Integrationstests müssen nur noch die **korrekte Kooperation** prüfen (Strukturtest).

Design for Testability – Kompositionalität

Herstellung von Kompositionalität im Design – Kompositionalität bzgl. Datentransformation und Kausalität:

- Zugriff auf globale Daten kapseln
- Zugriff auf kritische Abschnitte in gleichförmiger Art synchronisieren:
  - Verwendung von Monitoren
  - Verwendung von Semaphoren
  - Verwendung von Active-Wait-Mutex Verfahren

Achtung: bei Kompositionalität bzgl. Timing-Eigenschaften müssen ggf. parallele Komponenten auf mehrere Prozessoren verteilt werden!

Design for Testability – Strukturerhaltende Transformation von Anforderungen ins Design

- **Strukturerhaltende Transformation:** Struktur und Datenfluß der Anforderungsspezifikation wird für das Architekturdesign übernommen $\Longrightarrow$ direkte Beziehung zwischen Systemkomponente und der von ihr implementierten funktionalen Anforderung.

Vorteil der strukturerhaltenden Transformation beim Testen:

- **Überdeckung der funktionalen Anforderungen beim Testen erzeugt gleichzeitig bereits die Strukturüberdeckung.**

Design for Testability – HW/SW Testschnittstellen

Zielsetzung: Bereits im Design "Testpunkte", d.h. zusätzliche Schnittstellen für den späteren Test einplanen – am besten als permanente Systemschnittstellen vorsehen. Anwendungsbereiche:

- Monitoring der erzielten Überdeckung
- Künstliche Herbeiführung von im operationellen Betrieb selten auftretenden Zuständen
- Fehlerinjektion auf Software und Hardwareebene

Design for Testability – Kohäsion

- **Kohäsion:** SW-Komponenten tragen zur Implementierung von nur einer funktionalen Anforderung bei – es werden nicht mehrere funktionalen Anforderungen gleichzeitig in derselben Komponente bearbeitet.

Vorteile beim Testen:

- Reduktion der Anzahl erforderlicher Testfälle

Design for Testability – Separation von Kontrolle und Datentransformation

- **Separation von Kontrolle und Datentransformation:**
  - Kontrolle wird von übergeordneten Komponenten ausgeübt, die keine Datentransformationen durchführen.
  - Datentransformationen werden von untergeordneten Komponenten ausgeführt, die möglichst linear arbeiten und von den Kontrollkomponenten aktiviert werden.

Design for Testability – Separation von Kontrolle und Datentransformation

Vorteile beim Testen:

- Für den Test der Kontrollkomponenten kann von Nutzdaten weitgehend abstrahiert werden.
- Für die Erzielung der erforderlichen (Verzweigungs-)Überdeckung bei Datentransformationen sind weniger Testfälle erforderlich.

Design for Testability – Kapselung der kritischen Funktionen, Partitionierung

- **Partitionierung:** Trennung kritischer Funktionen von unkritischen Funktionen im Systemdesign – Ausführung der unkritischen Funktionen muß nebenwirkungsfrei für die kritischen Funktionen sein.
- **Kapselung:** Implementierung der kritischen Funktionen in möglichst wenigen Komponenten.

Design for Testability – Kapselung der kritischen Funktionen, Partitionierung

Aus Partitionierung und Kapselung resultiert verminderter Testaufwand:

- Unkritische Funktionen können mit geringerer Überdeckung geprüft werden. ($\Longrightarrow$ Das gilt nur bei guter Partitionierung – sonst kann ein Dominoeffekt von unkritischen zu kritischen Fehlern erzeugt werden!)
- Nur wenige Funktionen sind mit maximaler Überdeckung zu prüfen.
- Kein Integrationstest für kritische Funktionen im Zusammenspiel mit unkritischen erforderlich.

Design for Testability – Probleme beim OO-Design

Folgende OO-Merkmale erschweren den vertrauenswürdigen Test von OO-Komponenten:

- **Dynamische Objekterzeugung/Objektvernichtung:** Das korrekte Verhalten hängt nicht allein von der Programmierung der Objekte, sondern von den Zeitpunkten der Objektallokation/-deallokation ab. (**Analoges Problem:** Polymorphie und Vererbung mit dynamischer Bindung)

- **Generische Klassen:** Die Korrektheit des Codes für die generische Klasse garantiert noch nicht die Korrektheit jeder möglichen Instanz, da die konkreten Datentypen der Instantiierung besondere Probleme erzeugen können, die auf Klassenebene nicht sichtbar sind.

Anhang: Begriffe

Analytische Qualitätssicherung Sicherung der Qualität durch Untersuchung der Produkteigenschaften

Prozeßbezogene Qualitätssicherung Sicherung der Qualität durch Prüfung der Angemessenheit und korrekten Ausführung des Entwicklungsprozesses

Testen analytische Qualitätssicherung von ausführbaren SW-Programmen oder vollständigen HW/SW-Systemen

Dynamisches Testen Prüfung von ausführbaren SW-Programmen oder vollständigen HW/SW-Systemen durch dynamische Ausführung mit speziellen Eingabedaten

Statische Analyse analytische Qualitätssicherung von Software durch Untersuchung des Quellcodes ohne seine dynamische Ausführung

Validation Prüfung von Anforderungsspezifikation hinsichtlich Adäquatheit, Vollständigkeit, Widerspruchsfreiheit, Eindeutigkeit

Verifikation Prüfung eines Produktes gegen eine Referenzspezifikation

Formale Verifikation Prüfung eines Produktes gegen eine formale Spezifikation mittels mathematischer Nachweisverfahren