

Software- Qualitätsmanagement

Kernfach Angewandte Informatik

Sommersemester 2007

Prof. Dr. Hans-Gert Gräbe

<http://bis.informatik.uni-leipzig.de/HansGertGraebe>

Eine **Anomalie** ist jede Abweichung bestimmter Eigenschaften eines Programms von der korrekten Ausprägung dieser Eigenschaften.

- Konstruktive Sprachkonzepte erlauben das Aufdecken von Anomalien durch statische Quelltextanalyse
Beispiel: Typprüfung durch den Compiler
Oft ist keine unmittelbare Fehlererkennung, jedoch eine Identifikation von Fehlerort und -symptomen möglich.

Datenflussanomalien-Analyse

- **Ziel:** Aufdecken von Datenflussanomalien durch Analyse von Programmpfaden auf sinnvolle Datenfluss-Sequenzen

Datenfluss-Eigenschaften von Variablen

- Auf eine Variable x kann entlang eines Programmpfades wie folgt zugegriffen werden:
 - x wird definiert (d),
 - x wird referenziert (r),
 - x wird undefiniert (u) (z.B. beim Verlassen einer Methode)
 - x wird „geleert“ (e), d.h. der Wert an einen anderen Ort übertragen
- Enthält die Sequenz Teile, die keinen Sinn ergeben, so liegt eine Datenfluss-Anomalie vor.
Beispiele:
 - **rdru** die Sequenz beginnt mit einer Referenz, vor der Definition, diese Anomalie ist vom Typ **ur**
 - **ddrdu** diese Sequenz beginnt mit einer doppelten Definition (Anomalientyp dd) und endet mit **du**

Beispiel

```
/* swap (int a, int b) */  
int hilf; a=hilf; a=b; hilf=b;
```

Analyse:

a d : d d : r

b d : r r : r

hilf u : r d : e

Anomalietyp:

mehrfach nacheinander überschrieben

nie verändert

neu definiert vor e

```
/* swap (int a, int b) korrigierte Version */  
int hilf; hilf=b; b=a; a=hilf;
```

a d : r d : r

b d : r d : r

hilf u : d r : e

Leistung:

entdeckt Wertzuweisungen an falsche Variablen, Anweisungen an unkorrekter Stelle und fehlende Anweisungen

- **Vorteile:**

sichere Entdeckung bestimmter Fehlertypen,
geringer Aufwand im Vergleich zu dynamischen Verfahren,
direkte Fehlerlokalisierung und
gute Ergänzung zu anderen Testverfahren

- **Nachteile:**

Leistungsfähigkeit auf schmalen Fehlerbereich begrenzt

Analysierende Verfahren – Zusammenfassung

- Quellcodeanalyse
 - Bindung und Kopplung als qualitatives Maß für „guten Code“
 - Bindung auf der Ebene einzelner Funktionen
 - Bindung auf der Ebene von Datenabstraktionen
 - Bindung auf der Ebene von Datenmodellen
 - Bindung in Vererbungshierarchien
- Komponentenanalyse
 - Umfangsmetriken (Beispiel: Halstead-Metrik)
 - Strukturmetriken (Beispiel: McCabe-Metrik)
 - Bindungsmetriken
 - welche Funktion wird wo und wie oft aufgerufen?
 - OO-Metriken
- Anomalienanalyse

Systemqualität

- Die Produktqualität eines SW-Produktes wird durch die Qualität seiner Komponenten (**Komponentenqualität**) und deren Beziehungen untereinander (**Systemqualität**) bestimmt.
- Systemqualität ergibt sich wieder aus dem Wechselspiel von konstruktiven und analytischen Maßnahmen
 - durch konstruktive Maßnahmen werden Defekte von Anfang an vermieden
 - durch analytische Maßnahmen werden Defekte aufgedeckt und beseitigt
- Vor der Sicherung der Systemqualität steht die Sicherung der Qualität der beteiligten Komponenten

Überprüfung der Systemqualität erfolgt, zeitlich nacheinander, in drei Teststufen:

1. Integrationstest

Test der Bindungen zwischen den Komponenten

Vergleichbar mit Strukturtest in Komponenten

System wird als White Box betrachtet

2. Systemtest

Test des Systems als Ganzes gegen die Spezifikation

Vergleichbar mit Funktionstest in Komponenten

System wird als Black Box betrachtet

3. Abnahmetest

Systemtest in der Anwendungsumgebung und unter Beteiligung des Auftraggebers

endet mit formalen Abnahmeverfahren

Ziel:

fehlerfreies Zusammenwirken der System-Komponenten
überprüfen.

Voraussetzung:

Jede Systemkomponente muss vorher für sich allein getestet
worden sein.

Generelle Vorgehensweise:

System-Komponenten werden schrittweise zusammengesetzt
mit regelmäßiger Überprüfung auf Fehler nach jeder neuen
Komponente.

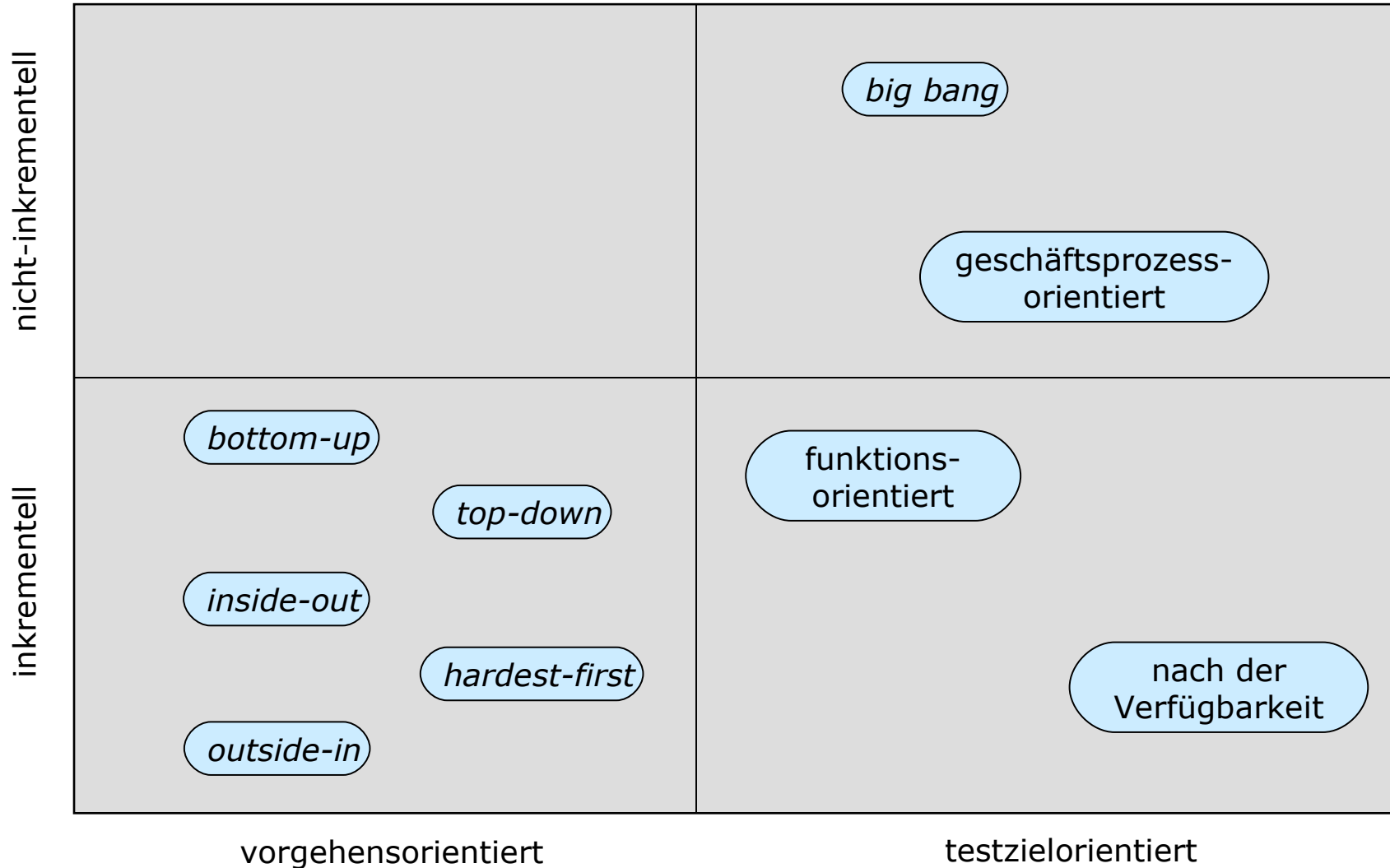
Vorgehensweise (**Integrationsstrategie**) hängt vom
verwendeten Entwicklungsmodell ab.

Integrationsstrategien

- zeitliche Reihenfolge, in welcher fertiggestellte und überprüfte Systemkomponenten zu einem Gesamtsystem integriert werden.
- Unterscheidung zwischen inkrementellen und nicht inkrementellen Strategien
- Unterscheidung zwischen vorgehensorientierten und testzielorientierten Strategien

Integrationstest

- Testaktivität, welche begleitend zur Integration das korrekte Zusammenwirken der einzelnen Systemkomponenten überprüft.
 - erfolgt parallel zur Systemintegration
 - verwendet weitgehend modifizierte Überprüfungsverfahren, die auch für den Komponententest eingesetzt werden
 - spezielle Testschnittstellen: Platzhalter und Testtreiber



Verschiedene Integrationsansätze

nicht-inkrementelle Integrationsstrategien

sehr viele oder sogar alle System-Komponenten (evtl. aus einem Teilbereich) werden gleichzeitig integriert

Vorteil: keine Platzhalter oder Testtreiber nötig

Nachteile:

- alle System-Komponenten müssen zur Integration zur Verfügung stehen
- Fehler sind schwer zu lokalisieren
- Testüberdeckung schwierig zu realisieren, da geeignete Testfälle schwer zu konstruieren sind

Beispiel geschäftsprozessorientierte Integration:

- Integration derjenigen Komponenten, die zu einem Geschäftsprozess gehören

Beispiel big bang:

- unsystematisch alles auf einmal

inkrementelle Integrationsstrategien

die System-Komponenten werden einzeln oder in kleinen Gruppen integriert

(Noch) nicht verfügbare Komponenten werden durch Testtreiber oder Platzhalter ersetzt

Vorteile:

- die Komponenten können integriert werden, sobald sie fertig sind
- leicht konstruierbare Testfälle, Testüberdeckung sichergestellt

Nachteil: u. U. sind viele Testtreiber oder Platzhalter nötig

Beispiel testzielorientierte Integrationsstrategien:

- Testfälle werden anhand der Testziele erstellt, etwa Testziel „frühzeitige Integration fertiger Komponenten“
- zur Überprüfung dieser Testfälle baut man dann die dafür benötigten System-Komponenten zusammen

verschiedene Vorgehensweisen sind möglich

- top-down, bottom-up, hardest-first, ...

Testtreiber und Platzhalter

Testtreiber (*driver*)

Spezielle Testschnittstelle, um System-Komponenten zu testen, deren Dienste nicht direkt von der Benutzungsoberfläche aufgerufen werden können.

Beispiel: Funktionalitätstest einer Datenbankbindung.
Testtreiber liefert die Eingaben und prüft die Antworten

Platzhalter (*dummies, stubs*):

Vorläufiger Ersatz für eine andere Komponente, die für den Test einer Systemkomponente benötigt wird, im Moment des Tests aber noch nicht verfügbar ist.

Beispiel: Fehlende Komponente soll Inventurliste ausdrucken.
Platzhalter übernimmt das, aber ohne Formatierung.

Im Allgemeinen sind Testtreiber leichter zu realisieren als Platzhalter.

top-down

Prüfung der System-Komponenten beginnend von der Wurzel der Baum- oder Schichtenhierarchie

schrittweise Integration, fehlende System-Komponenten werden simuliert

Vorteile:

- Frühzeitiges Simulationsmodell führt aus Sicht des Benutzers bereits einen Teil der Funktionen des endgültigen Systems aus.
- Änderungen, Alternativen, Verbesserungen frühzeitig sichtbar.
- Gezielte Prüfung der Fehlerbehandlung bei fehlerhaften Rückgabewerten möglich, da Rückgabewerte aus Platzhaltern stammen.
- Verzahnung von Entwurf und Implementierung ist möglich.

Nachteile:

- Platzhalter sind nötig (zusätzlicher Erstellungsaufwand).
- Bei zunehmender Integrationstiefe steigt die Schwierigkeit, bestimmte Testsituationen zu erzeugen.
- Zusammenwirken von zu prüfender Software, Systemsoftware und Hardware wird sehr spät untersucht.

bottom-up

Zuerst werden die Basiskomponenten integriert, da diese keine Dienste anderer Komponenten benötigen.

- Bei einer Baumhierarchie fängt man somit bei den Blättern an

Vorteile:

- keine Platzhalter nötig, leicht herstellbare Testbedingungen
- Testergebnisse sind leichter zu interpretieren
- bewusste Fehleingaben zur Prüfung der Fehlerbehandlung möglich
- Zusammenwirken von Systemsoftware, Hardware und zu prüfender Software wird früh getestet

Nachteile:

- Testtreiber erforderlich
- Fehler in der Produktdefinition werden erst spät gefunden, da lauffähiges Gesamtsystem erst am Ende verfügbar
- gezielte Überprüfung der Fehlerbehandlung für Rückgabewerte ist kaum realisierbar, da die realen Komponenten benutzt werden

outside-in

Kombination aus top-down und bottom-up, um die Vorteile beider zu vereinen und die Nachteile zu minimieren

man beginnt gleichzeitig von oben sowie von unten und arbeitet zur Mitte hin

inside-out

dieselbe Überlegung, aber man beginnt mit den System-Komponenten in der Mitte der Hierarchie und arbeitet nach oben und nach unten

vereint eher die Nachteile von *top-down* und *bottom-up*, daher nur u. U. mit hardest-first einsetzen

hardest-first

zuerst werden die kritischen, d.h. potenziell fehlerhaften und am schwierigsten zu implementierenden System-Komponenten implementiert und getestet

damit wird diese Komponente besonders oft getestet

Integrationstests

Die Integrationstests dienen der Überprüfung der **Schnittstellen** zwischen den System-Komponenten.

- Aufruf von Operationen, Funktionen und Prozeduren mit und ohne Parameterübergabe

- Verwendung von globalen Variablen oder Dateien

- Benutzung von global vereinbarten Konstanten und Typen

- Analyse der Bindung zwischen den Komponenten

Integrationstests werden unter verschiedenen Aspekten ausgeführt

Dynamischer Integrationstest

Ziel: Plausibilität der funktionalen Korrektheit durch Testfälle, die ausschließlich in der Integrationsphase nachweisbare Fehler abdecken.

Vorgehen: Stichprobentest, wie beim funktionalen Test von Komponenten

Strukturorientierter Integrationstest

wie Strukturtests für Komponenten

Kontrollflussorientierter Integrationstest betrachtet die unterschiedlichen Aufrufbeziehungen (Exporte und Importe) zwischen Komponenten.

Mögliche Überdeckungskriterien:

- jeder Aufruf jeder exportierten Operation muss in jeder importierenden Komponente wenigstens einmal überdeckt sein.
- alle Aufrufstellen sind in allen möglichen Reihenfolgen (mit Schleifenbeschränkung) zu überdecken.

Datenflussorientierter Integrationstest betrachtet die Programmstellen genauer, an denen importierte Operationen aufgerufen werden.

- analog datenflussorientierten Tests von Komponenten

Funktionaler Integrationstest

Prüft die spezifizierte Funktionalität der einzelnen System-Komponenten und deren Zusammenwirken.

Abweichungen liegen vor, wenn die Operation:

- zu wenig Funktionalität liefert (z. B. fehlende Teilfunktion),
- zu viel Funktionalität liefert (z. B. Aufruf einer unerwarteten Teilfunktion) oder
- falsche Funktionalität liefert.

Diese Fehler resultieren meist aus ungenauen Spezifikationen.

- werden beim Komponententest nicht erkannt, da dort die Spezifikation als „gesetzt“ gilt

Wertbezogener Integrationstest

Schnittstellen werden mit möglichst extremen Werten getestet

Entspricht der Grenzwertanalyse

Statischer Integrationstest

Analysierendes Verfahren des Quellcodes der beteiligten System-Komponenten

untersucht die Kopplung zwischen den Komponenten und erfasst Parameter systemweiter Metriken

- **Ziel:** zusätzliche und unnötige Kopplungen identifizieren und eliminieren

syntaktische Kompatibilität der Schnittstellen über die Komponentengrenze hinaus wird meist auf konstruktivem Weg erreicht

- automatische Überprüfung der Schnittstellendeklarationen durch den Compiler (Header-Dateien, Import-Deklarationen)

Analyse von Datenflussanomalien wie innerhalb von Komponenten

- **Verifizierende Methoden** können auch komponentenübergreifend eingesetzt werden.

Integration von OO-Systemen

Klassentest (Methodik bereits beschrieben)

Integration von einzelnen Methoden einer Klasse

Integrationstest der Vererbungen und Polymorphie

Integration von Unterklassen

Unterscheide Integration von reinen dienst anbietenden, reinen dienst nutzenden und gemischten Klassen

Situation: Oberklasse bereits integriert, Unterklasse für sich bereits getestet, Testmaterial für beide Tests liegt vor

neben Integrationstests für einzelne Methoden sind auch Integrationstests für Operationen und Folgen von Operationen erforderlich (geht wie bereits beschrieben)

Integration einzelner Methoden

- vollständig geerbte Methoden: neue Tests nicht erforderlich
 - beim Oberklassen-Integrationstest mit den Dienstnutzern bereits geprüft
 - redefinierte Methoden mit gleicher Semantik (Spezifikation): neue Testfälle nicht erforderlich
 - Integrationstests beziehen sich nur auf die Schnittstelle, funktionaler und struktureller Test erfolgte bereits beim Komponententest
 - redefinierte Methoden mit anderer Spezifikation: andere Testfälle erforderlich
 - Vor- und/oder Nachbedingungen restriktiver (Einsatzbereich spezieller)
- Bei restriktiveren Vorbedingungen Modifikation der Testfälle der Oberklasse erforderlich.
- Die alten Testfälle müssen auf die neue Zusicherung adäquat reagieren
 - Beispiel: allg. Konto -> Sparkonto
 - Typisch für dienst anbietende Klassen

Bei restriktiveren Nachbedingungen zusätzliche Testfälle erforderlich.

- zur Überdeckung neuer Nachbedingungen
- Beispiel: Speichern -> Speichern mit Rechteverwaltung
- Typisch für dienstnutzende Klassen.

Testen von Sequenzen

Oft Integration von ganzen Operationssequenzen erforderlich
Ereignisbaum-Methode

- ausgelöst durch ein Ereignis, Fortpflanzung durch Botschaften

Zerlegung in atomare Systemfunktionen

- Operationssequenzen werden durch ein Start-Ereignis angestoßen und durch ein Ausgabe-Ereignis abgeschlossen.