Software-Qualitätsmanagement

Vorlesung im Modul 10-202-2319 Software-Management

Sommersemester 2012

Prof. Dr. Hans-Gert Gräbe

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Umfangsmetriken – Die Halstead-Metrik

Misst die textuelle Komplexität eines Programms, indem die Zahl der verwendeten Funktionen und der verwendeten Variablen ins Verhältnis gesetzt werden.

- Es wird jeweils die Gesamtzahl (Programmtext) und die Zahl verschiedener Objekte (Symboltabelle) bestimmt.
- $\boldsymbol{\eta_1}$, $\mathbf{N_1}$ = Zahl der (verschiedenen) Funktionen, Operatoren, Symbole oder Schlüsselwörter (z. B.: +, -, *, /, **while**, **if**, ...)
- $\boldsymbol{\eta_2}$, $\mathbf{N_2}$ = Zahl der (verschiedenen) Variablen, Operanden ...

Interpretation:

- $\boldsymbol{\eta}=\eta_1+\eta_2$: Größe des Vokabulars
- $\mathbf{N}=N_1+N_2$: Länge der Implementierung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorteile:

- einfach zu ermitteln,
- bei jeder Programmiersprache verwendbar und
- gute Eignung der Metriken für die zu messenden Größen

Nachteile:

- nur der Implementierungsaspekt betrachtet und
- Mehrdeutigkeiten im Messansatz, z. B. bei den Klassifikationsregeln für Operatoren und Operanden

abgeleitete Größen:

- **D**= $\eta_1/2 \cdot N_2/\eta_2$,
  Parameter für die Schwierigkeit, den Code zu verstehen
- Interpretation:
  N_2/η_2 = durchschnittliches Vorkommen jeder Variablen,
  η_1 = Anzahl der verwendeten Funktionen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

```
int ZaehleVokale(String s) {
    int VokalAnzahl; char Zchn; int i;
    for(i=0; i < s.length(); i++) {
        Zchn=s[i];
        if  ((Zchn == 'A') || (Zchn == 'E') || (Zchn == 'I') ||
            (Zchn == 'O') || (Zchn == 'U'))  VokalAnzahl++;
    }
    return VokalAnzahl;
}
```

ZaehleVokale	1	**int**	3	()	9	++	2
VokalAnzahl	3	**String**	1	{}	2	[]	1
Zchn	7	**char**	1	;	8	==	5
s	3	**for**	1	=	2	||	4
i	5	**if**	1	<	1	.	1
Konstanten (6)	6	**return**	1	length	1		

$\eta_1=17, N_1=44, \eta_2=11, N_2=25, D=19.32$

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Kontrollfluss-Metriken – Die McCabe-Metrik

- Misst die strukturelle Komplexität eines Programms, indem eine Grapheninvariante, die **zyklomatische Zahl** V(G) des Kontrollflussgraphen, bestimmt wird.
- $V(G) = e - n + 2p$ mit
  - e = Anzahl der Kanten des Graphen
  - n = Anzahl der Knoten
  - p = Anzahl der Zusammenhangskomponenten
- Kontrollflussgraph wird für jede Prozedur aufgestellt (p=1).
- Für solchen Graphen gilt

$$\mathbf{V(G) = b + 1}$$

  mit **b** = Anzahl der Bedingungen.
- Zyklomatische Zahl ist additiv auf Komponenten.
- Lineare Teilstücke können zusammengezogen werden.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorteile:

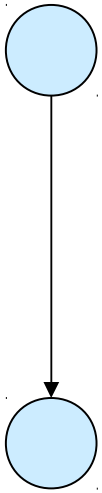
- einfach zu berechnen,
- grobes Maß für die Kontrollflusskomplexität: je größer, desto weiter weicht der Kontrollfluss vom linearen (V(G)=1) ab.

Nachteile:

- unterschiedliche Programmmerkmale werden stark vereinfacht
- Quellprogramm als zentrales Messobjekt überbetont
- Es wird nur das Programmgerüst, nicht aber die Komplexität einzelner und verschachtelter Anweisungen berücksichtigt

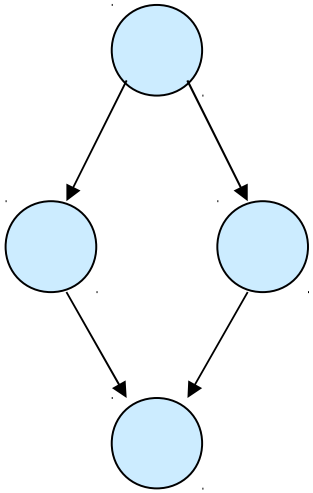
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Sequenz



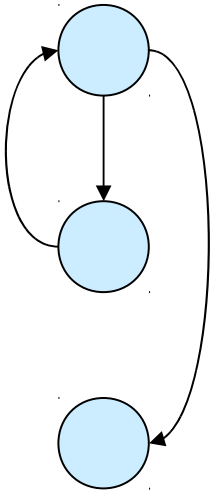
$V(G) = 1 - 2 + 2 = 1$

Auswahl

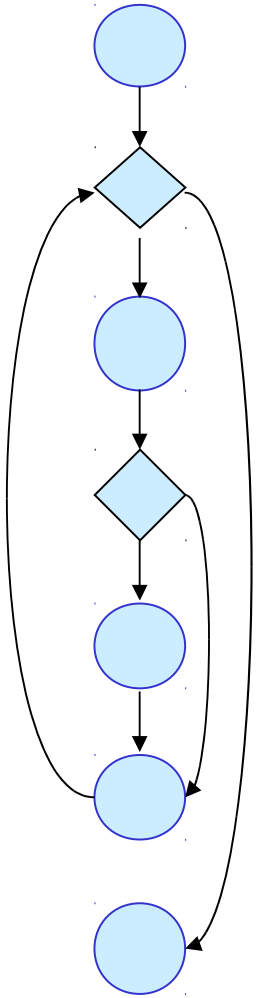


$V(G) = 4 - 4 + 2 = 2$

Abweisende Schleife



$V(G) = 3 - 3 + 2 = 2$



Das Programm ZaehleVokale hat die zyklomatische Zahl $V(G) = 8 - 7 + 2 = 3$

Es enthält zwei Bedingungen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Metriken für objektorientierte Entwicklung

Metriken der klassischen Software-Entwicklung sind in unveränderter Form für OO-Projekte nur bedingt aussagefähig.

- Umfangsmetriken: Wie mit geerbtem Code umgehen?
  - Durch Vererbung und Polymorphismus sinkt die Zeilenzahl signifikant
- McCabe-Metrik: Kontrollflusskomplexität bei OO meist sehr gering

Frage nach Maßen für die OO-spezifischen Effekte

- zusätzliche Maße waren erforderlich
  - Breite und Höhe der Vererbungshierarchie
  - Anzahl der Klassen, die eine spezielle Operation erben
  - Anteil wieder verwendeter Komponenten
  - Anzahl der Objekt- und Klassenattribute
  - Anzahl der Objekt- und Klassenoperationen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Typische Metriken (Beispiele)

Objekt- und Klassenattribute einer Klasse:

- Anzahl: |OV|, |CV|
- gewichtete Anzahl: $\sum$ T(v)
  - T(v) = Gewicht des Typs des Attributs v
  - gewichtet nach Zahl der Vorkommen, nach Zahl der Vorkommen in verschiedenen Methoden der Klasse etc.

Objekt- und Klassenmethoden einer Klasse:

- |OM|, |CM|, evtl. wieder gewichtet nach Komplexität
- Parameterkomplexität einer Methode
  - Zahl und Gewicht der Typen der Aufrufparameter
- McCabe-Metrik

durchschnittliche Komplexität von Klassen in einem Paket

- Durchschnittswerte der einzelnen Klassenparameter
- Kreuzreferenzparameter (Bindungsanalyse im Paket)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Folgende Metriken werden in der Literatur als signifikant genannt

DIT (*Depth of Inheritance Tree*):

- Tiefe des Vererbungsbaumes (Zahl der Vorfahren einer Klasse)
- je höher der Wert von DIT, desto höher die Fehlerwahrscheinlichkeit (Hoher Nachnutzungsgrad, Nichtbeachtung verdeckter Annahmen)

NOC (*Number of Children of a Class*):

- Anzahl der direkten Nachfolger einer Klasse
- je höher der Wert von NOC, desto geringer die Fehlerwahrscheinlichkeit (sehr präzise Abstraktion, Nichtvorhandensein verdeckter Annahmen)

RFC (*Response For a Class*):

- Anzahl der Funktionen, die direkt durch die Methoden einer Klasse aufgerufen werden.
- je höher der Wert von RFC, desto größer die Fehlerwahrscheinlichkeit (Hoher Delegationsgrad, Nichtbeachtung verdeckter Annahmen)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

WMC (*Weighted Methods per Class*):

- Anzahl aller neu definierten oder überschriebenen Methoden, die in jeder Klasse definiert sind.
- Geerbte Methoden werden nicht gezählt.
- je größer der Wert von WMC, desto höher die Fehlerwahrscheinlichkeit

CBO (*Coupling Between Object Classes*):

- eine Klasse ist mit einer anderen Klasse gekoppelt, wenn sie deren Methoden und/oder Attribute benutzt.
- CBO ist die Anzahl der Klassen, mit der eine Klasse gekoppelt ist.
- je größer der Wert von CBO, desto größer die Fehlerwahrscheinlichkeit (höherer Verschränkungsgrad)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Vorteile:

- erste Ansätze zur Verbesserung objektorientierter Komponenten
- erste empirische Untersuchungen zeigen die Eignung einiger Metriken als Qualitätsindikatoren

Nachteile:

- die Ziele der Metriken sind nur implizit erkennbar
  - Zielrelevanz der Metriken noch ungenügend untersucht
- keine Metriken für dynamische Aspekte
  - z. B.: Zustandsautomaten
- keine semantische Unterscheidung der Methoden
  - Standardoperationen (lesen, schreiben, erzeugen,...) sind weniger fehleranfällig als auszuprogrammierende "fachkonzeptspezifische" Methoden
- keine Berücksichtigung der Oberklassenqualität
  - ob eigene, geerbte oder fremde Methoden
  - ob Vererbung von Methoden aus Standardklassen oder eigenen
- keine Metriken, die eine "gute" Vererbungsstruktur prüfen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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** $\Rightarrow$ die Sequenz beginnt mit einer Referenz, vor der Definiton, diese Anomalie ist vom Typ **ur**
  - **ddrdu** $\Rightarrow$ diese Sequenz beginnt mit einer doppelten Definition (Anomalientyp **dd**) und endet mit **du**

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Beispiel

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

Analyse:		Anomalietyp:
a	d : d d : r	mehrfach nacheinander überschrieben
b	d : r r : r	nie verändert
hilf	u : r d : e	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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

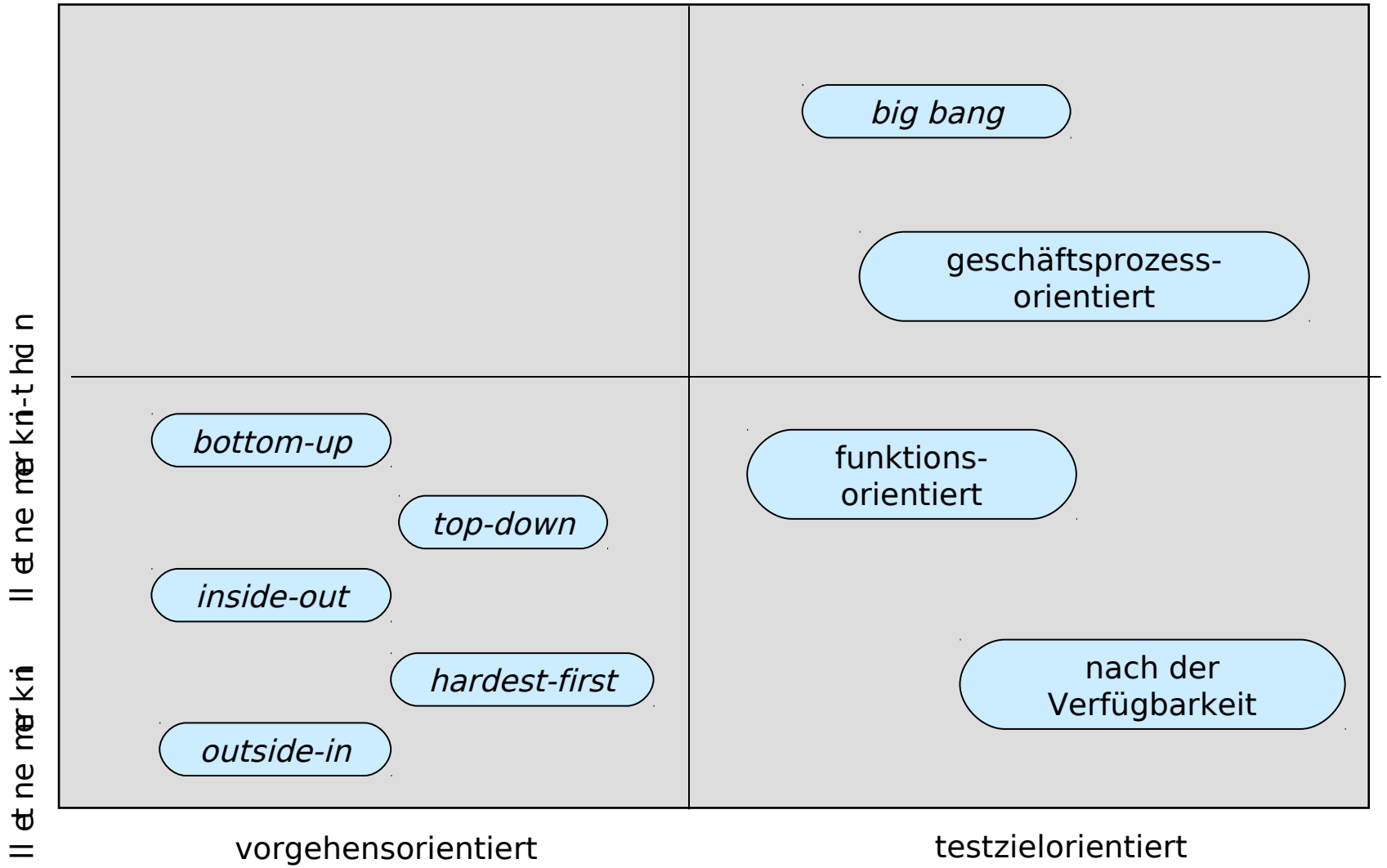
Integrationsstrategien

- 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 geschäftsprozess-orientierte Integration:
  - Integration derjenigen Komponenten, die zu einem Geschäftsprozess gehören
- Beispiel big bang:
  - unsystematisch alles auf einmal

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 testziel-orientierte 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, ...

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Testtreiber und Platzhalter

Testtreiber (*driver*)

- Spezielle Testschnittstelle, die eine zu testende Komponente aufruft und/oder steuert, deren Dienste nicht direkt von der Benutzungsoberfläche aufgerufen werden können.
- **Beispiel**: Funktionalitätstest einer Datenbankanbindung. Testtreiber liefert die Eingaben und prüft die Antworten

Platzhalter (*dummies, stubs*):

- Eine rudimentäre oder spezielle Implementierung einer Softwarekomponente, die verwendet wird, um eine noch nicht implementierte Komponente zu ersetzen bzw. zu simulieren, die im Moment des Tests 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 Rückgabe-werten 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 Komponenten implementiert und getestet
- damit wird diese Komponente besonders oft getestet

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 Kopplung 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Funktionaler Integrationstest

- Prüft die spezifizierte Funktionalität der einzelnen 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 und werden beim Komponententest nicht erkannt, da dort die Spezifikation als „gesetzt“ gilt.

Wertbezogener Integrationstest

- Schnittstellen werden mit möglichst extremen Werten getestet
- Entspricht der Grenzwertanalyse

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Statischer Integrationstest

- Analysierende 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
- Die *syntaktische* Kompatibilität der Schnittstellen über Komponentengrenze hinweg wird 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.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Der Systemtest

Der **Systemtest** ist der abschließende Test der Software-Entwickler und Qualitätssicherer in der realen Umgebung ohne den Auftraggeber.

- Umfasst Systemsoftware, Hardware, Bedienungsumfeld, technische Anlage
- System muss ggf. vor Beginn des Systemtests von der Entwicklungs- auf die Einsatz- oder Zielplattform portiert werden.
- **Basis**: Produktdefinition (Pflichtenheft, Produktmodell, Konzept der Benutzerschnittstelle, Benutzerhandbuch)
  - Pflichtenheft sollte sowohl die Qualitätsziele als auch die Testszenarien und Testfälle fixieren.
- Auf der Grundlage werden **Testfälle** aus den bisherigen Testzyklen übernommen und ergänzt.
- Zerlegung des Systemtests in verschiedene **Teiltests** an Hand zu bestimmender **Prüfziele**.
  - Prüfung aller geforderten Qualitätsziele in ihrer jeweiligen Ausprägung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Prüfziele

- Vollständigkeit
  - Sind alle funktionalen und nicht funktionalen Anforderungen aus dem Pflichtenheft erfüllt? (**Funktionstest**)
- Volumen
  - Systemtest mit umfangreichen Datenmengen (**Massentest**)
- Zeit
  - Systemtest auf Antwortzeiten unter starker Belastung (**Zeittest**)
- Zuverlässigkeit
  - Systemtest unter längerer Spitzenlast im geforderten „grünen“ Bereich (**Lasttest**)
  - auch unter Ausfall einzelner externer Hardware- oder Software-Komponenten
  - Mehrbenutzerbetrieb im Grenzbereich
  - Reaktion auf ungewöhnliche oder widersprüchliche Daten
- Robustheit und Fehlertoleranz
  - Systemtest unter Überlast, im „roten“ Bereich (**Stresstest**)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Benutzbarkeit
  - Test der Verständlichkeit, Erlernbarkeit, Bedienbarkeit aus der Sicht des Endnutzers (**Benutzbarkeitstest**)
  - Zielgruppenbezogen (Fachtermini, Metaphern etc.)
- Sicherheit
  - Datenschutzmechanismen, Zusammenspiel mit dem umgebenden System (**Sicherheitstest**)
- Interoperabilität
  - Relevant, wenn das System in einen größeren Verbund eingebettet ist (**Kompatibilitätstest**)
  - Kompatibilität der Schnittstellen und der Daten
- Konfiguration
  - wenn vorgesehen, Test der Systemausprägungen für verschiedene Hard- und Softwareplattformen (**Konfigurationstest**)
- Dokumentation
  - Vorhandensein, Angemessenheit und Güte der Benutzer- und Wartungsdokumentation (**Dokumentationstest**)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Teiltests

Funktionstest

- Test, ob alle in der Produktdefinition geforderten Funktionen vorhanden und wie vorgesehen realisiert sind.
- Testsequenzen sind aus dem Pflichtenheft zu übernehmen und/oder mit funktionalen Testverfahren systematisch und vollständig herzuleiten.

Leistungstest

- dient der Überprüfung des in der Produktdefinition festgelegten Leistungsverhaltens
  - Massentest, Zeittest, Lasttest, Stresstest
  - Einsatz eines Testdatengenerators oder realer Daten vom Auftraggeber oder von Pilotkunden
  - Frage der Systemstabilisierung nach Überlastphasen, etwa durch den Entzug von Ressourcen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Benutzbarkeitstest

- Oft entscheidend für die Akzeptanz eines Softwareprodukts
- Kann sehr aufwändig sein, wenn darauf in der Phase der Produktdefinition zu wenig Wert gelegt wurde

Interoperabilitätstest

- heutige Systeme sind in der Regel keine alleinstehenden Systeme, sondern in eine Standardumgebung integriert
  - umfasst meist eine komplexe GUI-Schnittstelle zum Betriebssystem
  - Frage der Interaktion mit diesen Oberflächen (etwa mit der Zwischenablage in Windows)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Installations- und Wiederinbetriebnahmetest

- **Installationstest**: Prüft, ob das System mit den erstellten Installationsbeschreibungen installiert und in Betrieb genommen werden kann.
- **Wiederinbetriebnahmetest**: Prüft, ob das System nach einer Unterbrechung oder einem Zusammenbruch des Basissystems mit den vorliegenden Beschreibungen wieder in Betrieb genommen werden kann und ob noch alle Daten aktuell und verfügbar sind.

Besonderheiten für OO-Systeme gibt es nicht, da der Systemtest ein Black-Box-Test ist, der gar nicht bemerken kann, ob das System ein OO-System ist.

Systemtest als Regressionstest: Aufzeichnen der Testfälle erlaubt es, diese bei späteren Fehlerkorrekturen oder inkrementeller Software-Entwicklung relativ problemlos zu wiederholen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Abnahmetest

Der **Abnahmetest** ist eine besondere Ausprägung des Systemtests, bei dem das System getestet wird

- unter Mitwirkung und Federführung des Auftraggebers
- in der realen Einsatzumgebung beim Auftraggeber
- (unter Umständen) mit echten Daten des Auftraggebers

Auftraggeber kann die Testfälle aus dem Systemtest übernehmen, modifizieren und eigene Testszenarien durchführen.

- Konzentration in der Regel auf den Test unter normalen Betriebsbedingungen
- Sollte bereits im Auftrag vereinbart sein, wird aber in der Regel ein „freies Testen“ sein.
- Verfahren des Abnahmetests sollte bereits beim Systemtest zum Einsatz kommen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Methodik aus Auftraggebersicht

- Erzeugen des zu testenden Systems aus den Quellen
  - hilfsweise Löschen aller Objektdaten
  - Bilden und Speichern einer Prüfsumme über das gesamte System, um dessen Unversehrtheit am Schluss zu prüfen
- Durchführung der Abnahme nach der vereinbarten Testvorschrift
  - Einbeziehung des Benutzerhandbuchs (mindestens alle dort angegebene Beispiele müssen funktionieren)
- regelmäßige einvernehmliche schriftliche Fixierung der Testergebnisse
- regelmäßiges freies Testen und Dokumentation dieser Testfälle
- Abnahme endet mit einer Schluss-Sitzung
  - Wichtung der protokollierten Fehler
  - Entscheidung über Annahmen, Auftrag zur Nachbesserung, Ablehnung

Abnahme stellt immer einen Kompromiss zwischen optimalem (also fehlerfreiem) und akzeptablem Ergebnis dar.