Software-Qualitätsmanagement

Vorlesung im Modul 10-202-2319 Software-Management

Sommersemester 2013

Prof. Dr. Hans-Gert Gräbe

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

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** ⇒ die Sequenz beginnt mit einer Referenz, vor der Definiton, diese Anomalie ist vom Typ **ur**
  - **ddrdu** ⇒ 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

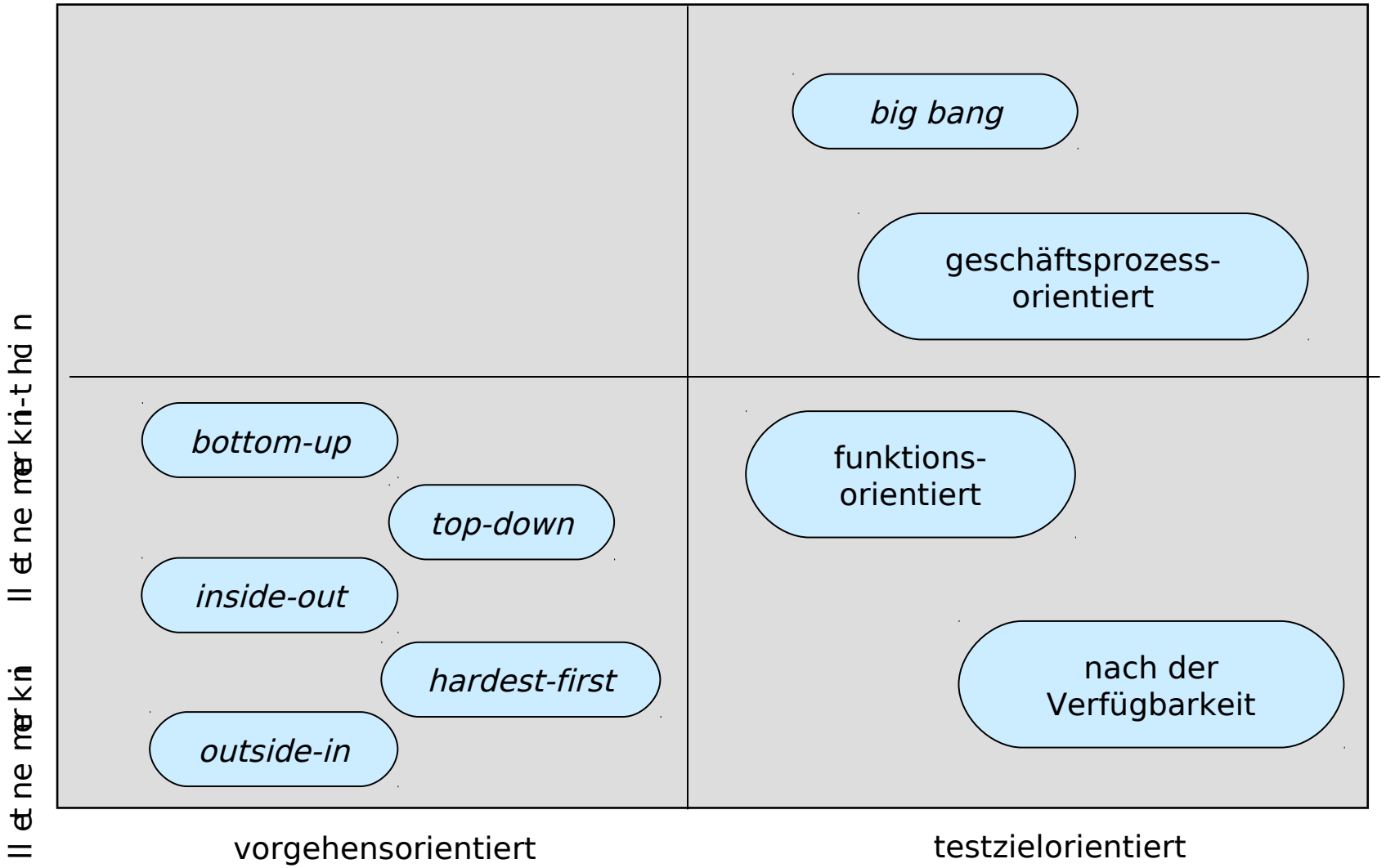
Integrationsstrategien

- 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.