

Software- Qualitätsmanagement

**Vorlesung im Modul 10-202-2319
Software-Management**

Sommersemester 2014

Prof. Dr. Hans-Gert Gräbe

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

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

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 zwischen Klassen der Komponente)

Folgende Metriken werden in der Literatur als signifikant für das Vermessen objektorientierter Klassenstrukturen 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)

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 Instanzen dieser Klasse 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).

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 Oberlassenqualität
 - ob eigene, geerbte oder fremde Methoden
 - ob Vererbung von Methoden aus Standardklassen oder eigenen
- keine Metriken, die eine "gute" Vererbungsstruktur prüfen

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** \Rightarrow die Sequenz beginnt mit einer Referenz, vor der Definition, diese Anomalie ist vom Typ **ur**
 - **ddrdu** \Rightarrow 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

1. Einführung und Überblick

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:

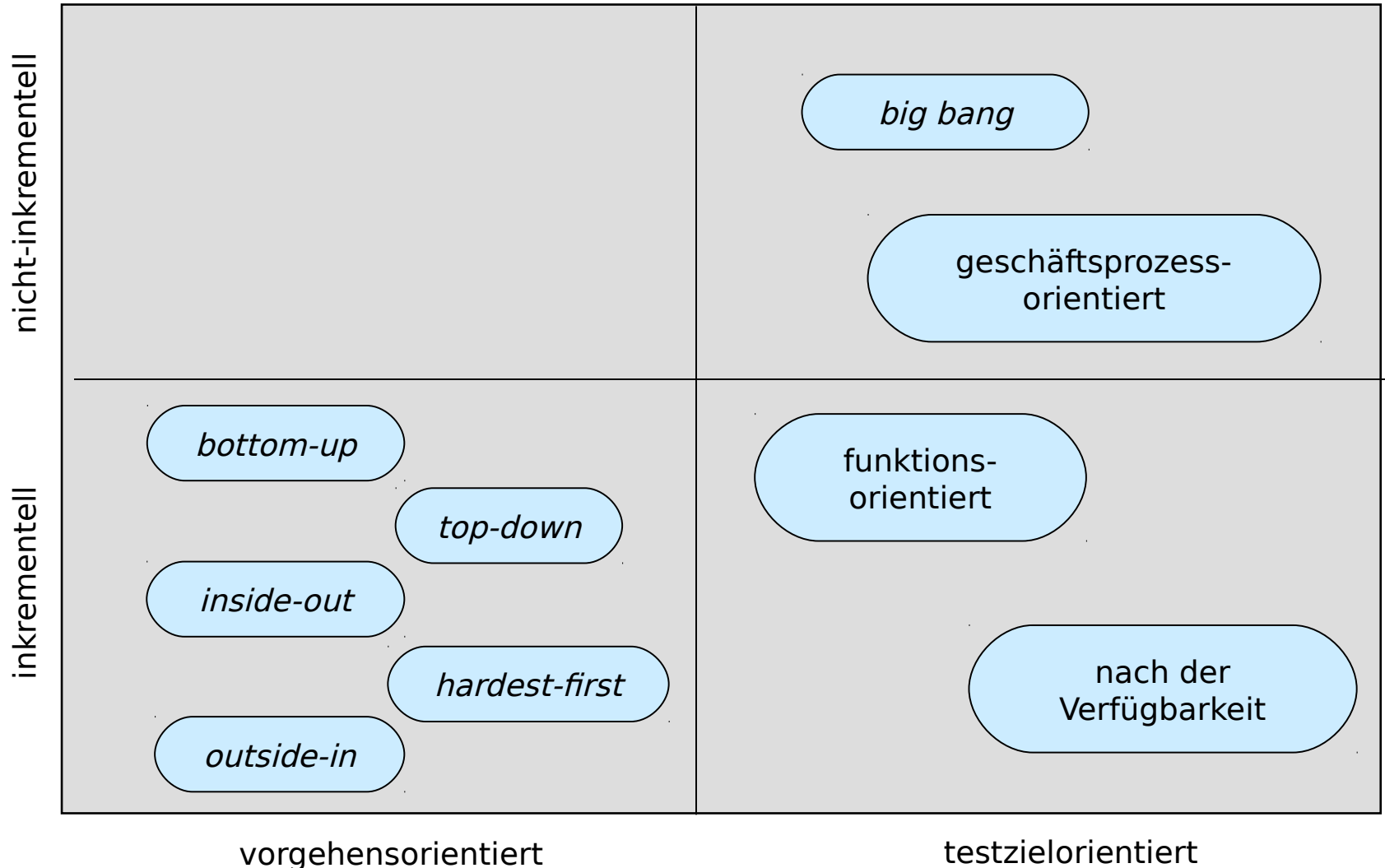
- System-Komponenten werden schrittweise zusammengesetzt mit regelmäßiger Überprüfung auf Fehler nach jeder neuen Komponente.
- Vorgehensweise (**Integrationsstrategie**) hängt vom verwendeten Entwicklungsmodell ab.

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 geschäftsprozess-orientierte 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 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, ...

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.

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 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 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

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 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

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.

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 Entwicklung 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

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**)

- 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**)

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

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)

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.

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

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.

Abnahme größerer Systeme

Mehrstufiges Abnahmeverfahren:

Werkabnahme

- Abnahme in einer speziellen werksseitig erstellten Testumgebung
- sinnvoll nur, wenn Installation weit entfernt erfolgen soll oder wenn die Installation den Betriebsablauf nachhaltig stört

Abnahme in der realen Umgebung

- unverzichtbar, evtl. sind Maßnahmen zur Sicherung des Betriebsablaufs zu treffen
- Durchführung auch der Tests, auf die in der Werksabnahme verzichtet werden musste, weil deren Implementierung in der Testumgebung zu aufwändig gewesen wäre

Betriebsabnahme

- Versuchsbetrieb in der Garantiephase mit aufwändigerer Protokollierung des Betriebs
 - Aufzeichnung aller Fehler, Ergänzung der Testreihe
 - Wiederholung der modifizierten Tests mit dem verbesserten System vor der endgültigen Inbetriebnahme

Abnahme von Produkten für den anonymen Markt

Auftraggeber und Nutzer sind verschieden.

- Interner Auftraggeber (Marketingabteilung, Produktmanager) nimmt das Produkt ab
- Systeme werden in der Regel einem Alpha- und Beta-Test unterzogen
 - Prüfziele Fehlertoleranz, Benutzbarkeit, Konfiguration und Interoperabilität lassen sich nur schwer durch den internen Auftraggeber testen
 - aufgetretene Fehler werden protokolliert und beseitigt
- **Alpha-Test:** System wird in der Zielumgebung des Herstellers durch Anwender erprobt.
- **Beta-Test:** System wird ausgewählten Pilot-Kunden in deren eigener Umgebung zur Probenutzung zur Verfügung gestellt.
 - nach umfangreichen Fehlerkorrekturen auch Beta2-Phase möglich
 - Pilotkunden erhalten beim späteren Kauf meist einen Rabatt

Beispiel: Roadmap für Windows Longhorn

(<http://www.winsupersite.com>)

- Developer Preview (April 2005)
- Beta 1 (Juni 2005) – parallel zur Integration
- Beta 2 (Ende 2005)
- RC0 (Febr. 2006) – Client Release Candidate, „status feature complete“
- RC1 (April 2006)
- RTM (Juni 2006) – Client release to manufacturing
- Launch (Oktober 2006) – allgemeine Verfügbarkeit
- Server RTM (Client RTM + 6 Monate)

Produktzertifikate

Die Qualität eines Software-Systems als *Produkt* ist für den Endkunden von eigenständigem Interesse. Hersteller sind damit an Zertifikaten für ihre Softwareprodukte durch neutrale Instanzen interessiert, die gewöhnlich durch Hersteller übergreifende domänen-spezifische *Gütegemeinschaften* als branchenspezifische Form der Selbstorganisation autorisiert sind.

- Erstmals 1985: Richtlinie der *Gütegemeinschaft Software* (als übergreifende Instanz von Software-Anbietern) zur einheitlichen Prüfung von Software-Produkten
 - Reine Produktnorm, keine Aussage über den Entwicklungsprozess
 - Qualitätsanforderungen beziehen sich auf Produktbeschreibung zu Informationen des Kunden vor dem Kauf, Dokumentation, Programme und Daten
 - Nicht berücksichtigt werden unterstützende Dienstleistungen

Produktzertifikate

- Überarbeitung als DIN 66285 sowie ISO 12119:1994 „Information technology – Software packages – Quality requirements and testing“
 - Applicable to software packages. Establishes requirements for software packages and instructions on how to test a software package against these requirements. Deals only with software packages as offered and delivered. Does not deal with their production process. The quality system of a supplier is outside the scope of this standard. (<http://www.iso.org>)
- ISO 25051:2006 „Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Requirements for quality of Commercial Off-The-Shelf (COTS) software product and instructions for testing“
 - Defines quality requirements for COTS software products and includes the product description, the user documentation, and the software contained on a computer sensible media.
 - The quality requirements, functionality, reliability, usability, efficiency, maintainability, portability, and quality in use are consistent with the definitions of SquaRE. (<http://www.iso.org>)

Produktzertifikate

- Aufgegangen in eine ganze Serie von Normen, die durch die ISO 25000:2014 „Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE“ genauer beschrieben wird
 - ISO/IEC 25000:2014 provides guidance for the use of the new series of International Standards named Systems and software Quality Requirements and Evaluation (SQuaRE). The purpose of ISO/IEC 25000:2014 is to provide a general overview of SQuaRE contents, common reference models and definitions, as well as the relationship among the documents, allowing users of the Guide a good understanding of those series of standards, according to their purpose of use. It also contains an explanation of the transition process between the old ISO/IEC 9126 and the ISO/IEC 14598 series and SQuaRE. (<http://www.iso.org>)
 - IEC = International Electrotechnical Commission, ISO/IEC JTC 001 – einziges gemeinsames Normungsgremium innerhalb der ISO (JTC = Joint Technical Committee)

Beispiel

(Quelle: <http://www.tuev-sued.de> „Prüfungen Softwarequalität“)

Wir prüfen Ihr Softwareprodukt (Internet, Client, Server, SaaS, ...) auf erfolgreiche Umsetzung der Anforderungen zur Funktionalität, Usability und Datensicherheit.

Die Grundlagen unserer Prüfungen stellen die gesetzlichen Vorschriften und die Vorgaben aus den relevanten anerkannten Standards dar.

- **Funktionalitäts-Prüfung:** Es werden die Anforderungen der ISO 25051: "Software-Engineering - Softwareproduktbewertung - Qualitätsanforderungen an kommerzielle serienmäßig produzierte Softwareprodukte (COTS) und Prüfanweisungen" angewendet.
- **Usability-Prüfung:** Für die Prüfung in ergonomischer Hinsicht wird die DIN EN ISO 9241 Ergonomie der Mensch-System-Interaktion - Teil 110: "Grundsätze der Dialoggestaltung" zu Grunde gelegt.
- **Datensicherheits-Prüfung:** Es werden die gesetzlichen Anforderungen (Bundesdatenschutzgesetz, Telemediengesetz), die BSI-Empfehlung von Standardsicherheitsmaßnahmen (IT-Grundschutz) sowie in der Praxis gängige Standards zu Grunde gelegt.

Qualitätssicherung in der Phase Betrieb und Wartung

Bug Tracking

- Einsatz automatisierter webgestützter Systeme wie Bugzilla

Arbeit mit Power Usern

- Rolle von Alpha- und Beta-Test-Phasen

Rolle der Qualität der Daten

ISO 9000 sieht Pflicht zur Nachweisführung vor.

- Wartungsplan, Wartungsaufzeichnungen und -berichte
- Konfigurationsmanagementplan