Softwarequalität und -test

2. Vorlesung

„Qualität kompakt“
(Zweiter Teil)

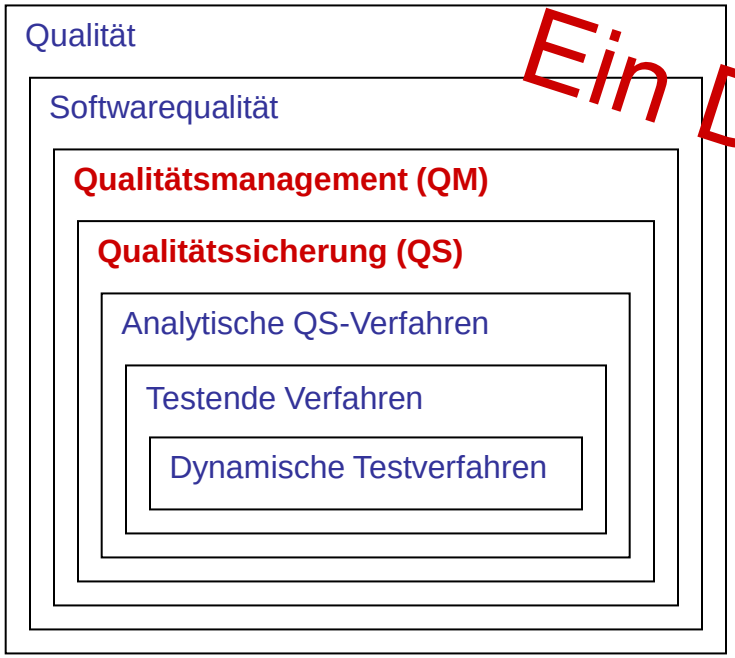
www.beuth-hochschule.de

Dipl.-Inform. Thomas Ziemer

Softwarequalität und -test

Qualität kompakt

Genereller Ansatz zur Beschreibung von Qualität



Ein Durchstich!

Softwarequalität und -test

Qualität kompakt

Qualitätsmanagement und Qualitätssicherung

Es genügt nicht, Qualitätsanforderungen nur aufzustellen. Dies geschieht mit Hilfe eines Qualitätsmanagements (QM). Genau so wichtig ist es, sicherzustellen, dass diese Qualitätsanforderungen auch erreicht werden! Dazu dient die Qualitäts-sicherung (QS).

Softwarequalität und -test

Qualität kompakt

Qualitätsmanagement und Qualitätssicherung (Fortsetzung)

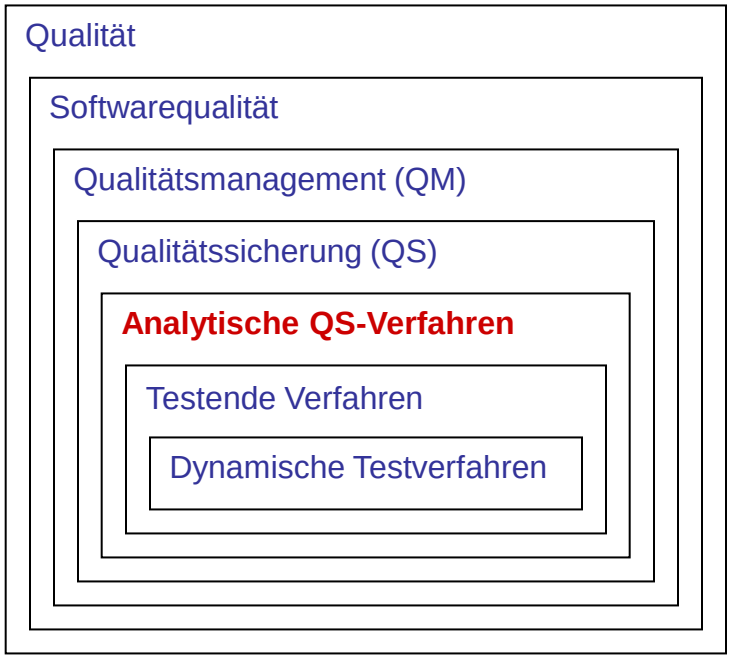
Def.: [Qualitätsmanagement umfasst alle…] *Tätigkeiten der Gesamtführungsaufgabe, welche die Qualitätspolitik, Ziele und Verantwortungen festlegen sowie diese durch Mittel wie Qualitätsplanung, Qualitätslenkung, Qualitätssicherung und Qualitätsverbesserung im Rahmen des Qualitätsmanagementsystems verwirklichen.*
(DIN EN ISO 8402)

Def.: [Qualitätssichernd sind…] *alle geplanten und systematischen Tätigkeiten, die innerhalb des Qualitätsmanagementsystems verwirklicht sind, und die wie erforderlich dargelegt werden, um angemessenes Vertrauen zu schaffen, dass eine Einheit die Qualitätsanforderung erfüllen wird.*
(DIN EN ISO 8402)

Softwarequalität und -test

Qualität kompakt

Genereller Ansatz zur Beschreibung von Qualität



Softwarequalität und -test

Qualität kompakt

Analytische Verfahren zur Qualitätssicherung

Analytische Qualitätssicherungsverfahren lassen sich nach verschiedenen Kriterien klassifizieren. In Abhängigkeit von der Zielsetzung lassen sich drei Klassen unterscheiden:

1. Testende Verfahren
   Sie haben das Ziel, Fehler zu erkennen:
   - Dynamische Testverfahren (die Systemkomponente wird ausgeführt, um Fehler zu finden)
   - Statische Testverfahren die Systemkomponente wird nicht ausgeführt, sondern der Quellcode analysiert, um Fehler zu finden (siehe auch „3. Analysierende Verfahren“ weiter hinten)
   –

Softwarequalität und -test

Qualität kompakt

Analytische Verfahren zur Qualitätssicherung (Fortsetzung)

2. Verifizierende Verfahren
   Sie beweisen die Korrektheit einer Systemkomponente:

   - Verifikation beweist mit mathematischen Mitteln [beispielsweise mittels Prädikatenkalkül] (siehe Hoare) die Konsistenz zwischen der Spezifikation und der Implementierung einer Systemkomponente)

Softwarequalität und -test

Qualität kompakt

Analytische Verfahren zur Qualitätssicherung (Fortsetzung)

3. Analysierende Verfahren

Sie vermessen bestimmte Eigenschaften von Systemkomponenten:

- Analyse der Bindungsart (ein funktionales Modul soll funktional gebunden sein; eine funktionale Bindung liegt vor, wenn alle Elemente des Moduls an der Verwirklichung einer **einzigen, abgeschlossenen Funktion** beteiligt sind)

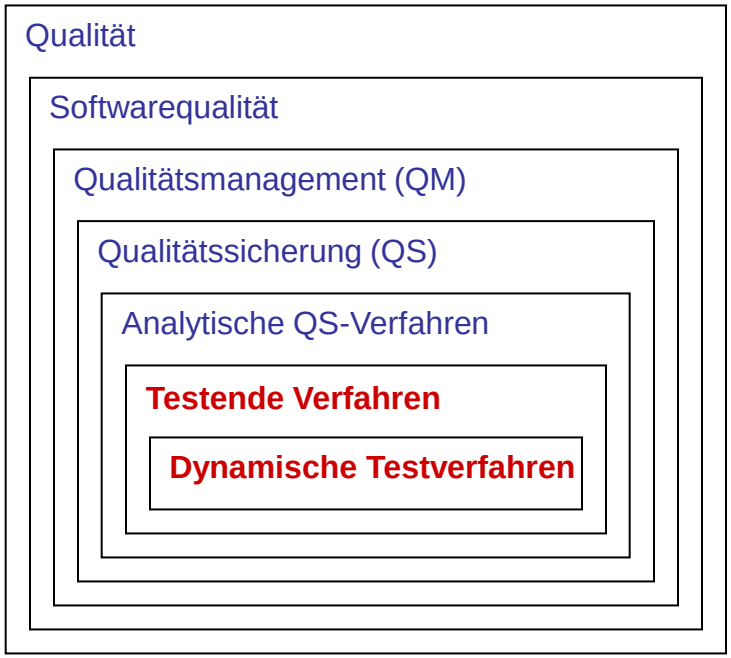
- Metriken (Eigenschaften wie die strukturelle Komplexität, die Programmlänge oder den Grad der Kommentierung quantitativ ermitteln und mit bisherigen Maßzahlen vergleichen)

- Grafiken und Tabellen (erlauben es, ein Programm unter speziellen Gesichtspunkten zu analysieren, und die Ergebnisse in gut lesbarer und interpretierbarer Form darzustellen)

Softwarequalität und -test

Qualität kompakt

Genereller Ansatz zur Beschreibung von Qualität



Softwarequalität und -test

Qualität kompakt

Dynamische Verfahren zur Qualitätssicherung

Dynamische Testverfahren, bei denen Fehler in Systemkomponenten durch die Ausführung des entsprechenden Codes identifiziert werden, können noch weiter klassifiziert werden:

Sie haben das Ziel, Fehler zu erkennen mittels:

- Zufallstests
- Test spezieller Werte
- Funktionale Äquivalenzklassenbildung
- Grenzwertanalyse
- Parallelalgorithmen
- Modultests

Testfälle

Qualität kompakt

Die richtigen Testfälle finden

Die richtigen Testfälle für eine Operation zu finden, ist häufig nicht einfach. Prüft man zu wenige oder die falschen Testfälle, wiegt man sich schnell in falscher Sicherheit – das ist mitunter gefährlicher, als gar nicht zu testen!

Sinnvoll ist es, typische Anwendungsfälle zu testen; dafür nimmt man Werte, mit denen die Operationen tatsächlich im Programm aufgerufen werden. Einige Richtlinien helfen, die richtigen Testfälle zu identifizieren:

Qualität kompakt

Die richtigen Testfälle finden (Fortsetzung)

- Für jeden von einer Operation akzeptierten Datentyp gibt es spezielle Werte, die es zu betrachten lohnt: bei ganzzahligen Typen wie *int* und *long* sind es etwa *0*, *1*, *2*, ihre negativen Werte sowie die Konstanten *MAX_VALUE* und *MIN_VALUE*.

- Für String-Parameter sollte man (wie auch bei anderen Objekttypen) überprüfen, ob die Operation auf die Übergabe von *null* wie vorgesehen reagiert.

- Auch der Leerstring ist immer „einen Test wert“, ebenso ein String mit nur einem Zeichen, mit einem Steuerzeichen wie *n* und ein sehr langer String.

–

Grenzwertanalyse

Qualität kompakt

Äquivalenzklasse und Grenzwertanalyse

Die Definitionsbereiche der Eingabeparameter und die Wertebereiche der Ausgabeparameter (des Ergebnisses) werden in Äquivalenzklassen zerlegt.

- Es wird davon ausgegangen, dass ein Programm bei der Verarbeitung eines Repräsentanten aus einer Äquivalenzklasse so reagiert wie bei allen anderen Werten aus dieser Klasse.

- Wenn das Programm mit dem repräsentativen Wert aus der Äquivalenzklasse fehlerfrei läuft, dann ist zu erwarten, dass es auch für andere Werte aus dieser Klasse korrekt funktioniert.

–

Äquivalenzklasse und Grenzwertanalyse (Fortsetzung)

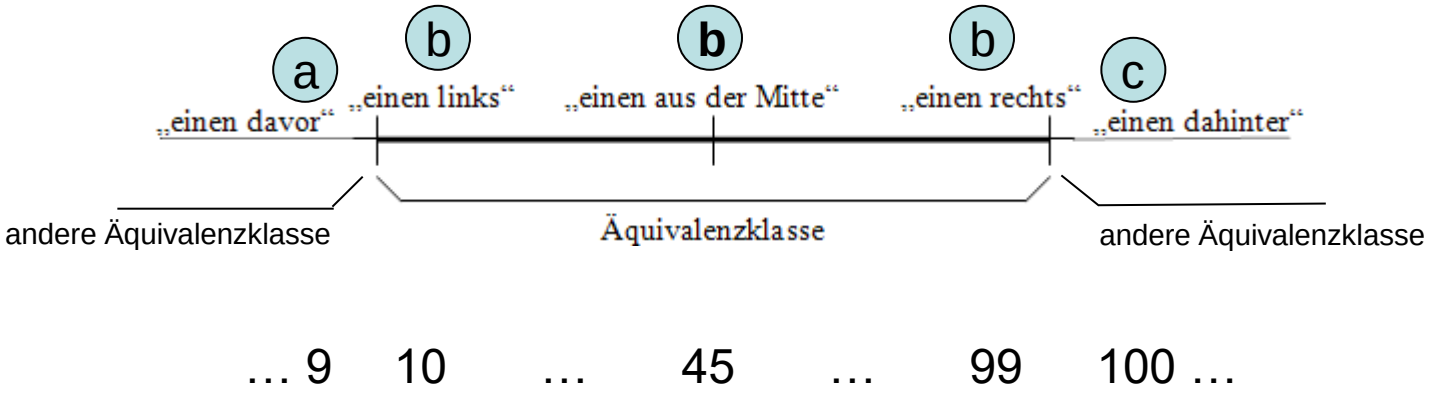
- Erfahrungen haben gezeigt, dass Testfälle, die die Grenzwerte der Äquivalenzklassen abdecken oder in unmittelbarer Umgebung dieser Grenzen liegen, besonders effektiv sind, d.h. besonders häufig Fehler aufdecken.

  Sinnvolle Testfälle wählt man demnach aus den Grenzen dieser Bereiche aus.

Qualität kompakt

Äquivalenzklasse und Grenzwertanalyse (Fortsetzung)

Als Faustregel zur Bildung der Grenzwerte gilt im allgemeinen: *„einen links, einen rechts, einen davor, einen dahinter und einen aus der Mitte“*. Das sind insgesamt fünf Testfälle, durch die eine Äquivalenzklasse abgedeckt wird:



Modultest

Softwarequalität und -test

Qualität kompakt

Modultest

Mit Modultests (*Unit Tests*, z.B. *JUnit*) kann schon während der Entwicklung in der Implementierungsphase immer wieder überprüft werden, ob ein Programm wie vorgesehen arbeitet. Modultests zeichnet aus:

- Finden von Fehlern in Modulen, die schon längst funktionieren sollten
  Minimierung des Debuggens durch Regressionstests

- Fehlersuche automatisieren und wesentlich beschleunigen
  Für jedes relevante Stück Code zunächst eine eigene Testroutine schreiben; Tests lassen sich danach beliebig oft wiederholen.

  –

Softwarequalität und -test

Qualität kompakt

Modultest (Fortsetzung)

- Modultests untersuchen stets einen möglichst kleinen, für sich allein funktionierenden Code-Ausschnitt, etwa eine einzelne Methode oder Klasse.

- Mit ihnen können bereits die Entwickler Detailfehler aufdecken und preiswert beseitigen, bevor die Testabteilung oder gar der Kunde darauf stößt.

–

Qualität kompakt

Parallelalgorithmus

Ein alternativer Algorithmus mit…

…ja, mit was denn?

Qualität kompakt

Parallelalgorithmus

Ein alternativer Algorithmus mit…

…(zumindest theoretisch erwarteten) identischen Ergebnissen dient zum parallelen Vergleich des zu testenden Algorithmus’.

Black-Box- und Glass-Box-Test

Black-Box-Tests (auch funktionale Tests genannt) verwenden den Ansatz, nur aufgrund der offengelegten Schnittstellen eines Testmoduls dieses mit Eingaben zu versorgen und das Ergebnis auszuwerten. Die Syntax und Semantik des der implementierten Methode zugrundeliegenden Algorithmus' wird nicht interpretiert.

Wer nicht den *Test First*-Ansatz verfolgt, und stattdessen die Tests für bereits existierende Operationen implementiert, kann Testfälle jedoch auch durch Analyse des Quellcodes identifizieren. Diese Vorgehensweise nennt man Glass-Box-Test.

–

Weitere Testverfahren

Trotz der Vorteile stößt der Entwickler häufig an die Grenzen von Modultests. Sie sind zwar unabdingbar, wenn Algorithmen getestet werden sollen, bieten jedoch keine Lösung, um komplexe grafische Benutzungsoberflächen zu testen, über die der Anwender schließlich auf die Algorithmen zugreift. Hierzu gibt es zusätzliche Testverfahren:

- GUI-Test
- Inhalts- und Pixelvergleich

Puh, fertig!