Systematisches Testen von Software

Markus Eckstein
Systematika Information Systems GmbH
Kurfürsten-Anlage 36
69115 Heidelberg
markus.eckstein@systematika.com

Zusammenfassung

Die wichtigsten Auswirkungen des systematischen Testens sind weniger Fehler, höhere Qualität und daher bessere Softwareprodukte und somit erfolgreiche Softwareprojekte. Um Software möglichst fehlerfrei zu erstellen, haben sich einige Vorgehensweisen und Verfahren in der Praxis bewährt. Systematisches Testen hilft Ihnen, keine Bananensoftware zu erstellen, die erst beim Kunden reifen muss, sondern hochwertige Software, die die Anforderungen erfüllt.

Schlüsselwörter: Softwaretesting, Modultest, Integrationstest, Systemtest, Blackbox-Whitebox-Tests, Testmetriken, Code-Coverage

1 Motivation

Komplexe Softwaresysteme sind in den seltensten Fällen 100% fehlerfrei. Beispielsweise ist durch einen Softwarefehler 1996 die europäische Trägerrakete Ariane 5 nach dem Start automatisch selbstzerstört worden, da ein Fehler bei der Berechnung der Geschwindigkeit vorlag. Hierdurch entstand ein Schaden in Höhe von ca. 500 Mio. €. Um Software möglichst fehlerfrei zu erstellen, haben sich einige Vorgehensweisen und Verfahren in der Praxis bewährt. Die wichtigsten Auswirkungen durch systematisches Testen sind weniger Fehler, höhere Qualität und daher bessere Softwareprodukte und somit erfolgreiche Softwareprojekte.

2 Softwaretesting

2.1 Testziele und Testanforderungen

Primäres Ziel des Testvorgehens ist eine korrekte (fachliche) Funktionalität und weniger Fehler in der Software sowie die Einhaltung der Qualitätsrichtlinien wie Wartbarkeit, Modularität und das Einhalten der Layoutvorgaben. Insgesamt wird hierdurch die Qualität der Software nicht nur gesichert, sondern auch erhöht.

Die Tests sollen nicht nur ihre Testziele erreichen, sondern auch ausreichend dokumentiert werden. Hilfreich sind eine Wiederholbarkeit der Tests und deren Automatisierung. Insbesondere bei der Entwicklung eines neuen Releases kann hierdurch der Testauf-

wand verringert werden, da bestehende Tests wiederverwendet werden können oder die Grundlage der neuen Tests bilden.

2.2 Testmetriken

Ungenaue und vage formulierte Testziele erschweren deren Erreichung, da ein Maßstab fehlt, der die Zielerreichung messbar macht. Hier unterstützen Testmetriken, die einerseits die exakt formulierten Testziele messbar machen und deren Erfüllung die Testphase beenden. Als Maßstäbe der Tests bieten sich folgende Metriken an:

- Zeit für Fehlerfinden
- Anzahl gefundener Fehler / Anzahl Code-Zeilen
- Anzahl gefundener Fehler und deren Klassifikation in leicht, mittel und schwer
- Meldungen im SAS-Log
- Überdeckungsgrad

2.2.1 Meldungen im SAS-Log

Bei Meldungen im SAS-Log kann das Kriterium anstatt „keine Error“ auch strenger definiert werden und zusätzlich gefordert werden, dass SAS-Warnings und manche SAS-Notes ebenfalls nicht auftreten dürfen. Zum Beispiel

- `WARNING: The data set ... may be incomplete. ...`
- `WARNING: The variable ... in the DROP, KEEP, or RENAME list has never been referenced.`
- `NOTE: Character values have been converted to numeric values at ...`
- `NOTE: Invalid numeric data, ... = ... , at ...`
- `NOTE: Missing values were generated as a result of performing an operation on missing values....`

2.2.2 Überdeckungsgrad, Coverage

In vielen Programmiersprachen sind Tools zur Messung der Code-Überdeckung weit verbreitet. Bei der Code-Überdeckung wird gemessen, welche Code-Teile, insbesondere Statements, Zweige oder Pfade, bei der Durchführung der Tests ausgeführt werden. Je nach Überdeckungsgrad ändert sich die Anzahl der benötigten Testfälle.

An den einfachen SAS-Code

```
if (A > B or C > D) then ERG = "ok";
```

ergeben sich folgende mögliche Testdaten für die unterschiedlichen Überdeckungen:

- C_0-Überdeckung: Alle Statements werden ausgeführt
  1) A > B; C, D beliebig
- C_1-Überdeckung: Alle Zweige werden ausgeführt
  1) A > B; C, D beliebig
  2) A < B, C < D
- C_2-Überdeckung: Bedingungsteile werden variiert
  1) A > B, C < D
  2) A < B, C > D
- C_3-Überdeckung: Alle Kombinationen der Bedingungsteile werden ausgeführt
  1) A > B, C > D
  2) A < B, C < D
  3) A > B, C < D
  4) A < B, C > D
- C_∞-Überdeckung: Alle Pfade werden ausgeführt
  Bei diesem einfachen Beispiel entspricht dies der C_1-Überdeckung, da nur ein Pfad existiert. Im Allgemeinen sind die Testfälle für eine vollständige C_∞-Überdeckung sehr umfangreich.

2.3 Testarten

Man kann grundsätzlich zwei Arten des Tests unterscheiden, je nachdem ob die innere Struktur des Objekts für den Tester bekannt ist oder nicht.

2.3.1 Blackbox-Test

Bei dieser Testart ist die innere Struktur des Testobjekts nicht bekannt, daher wird mit dieser Testart funktionsorientiertes Testen durchgeführt. Die Testfälle werden gemäß der Funktionsbeschreibung (Spezifikation) erstellt. Methoden hierfür sind die Bildung von Äquivalenzklassen und die Grenzwertanalyse. Zusätzliche Implementierungen (Computerkriminalität) werden nicht entdeckt.

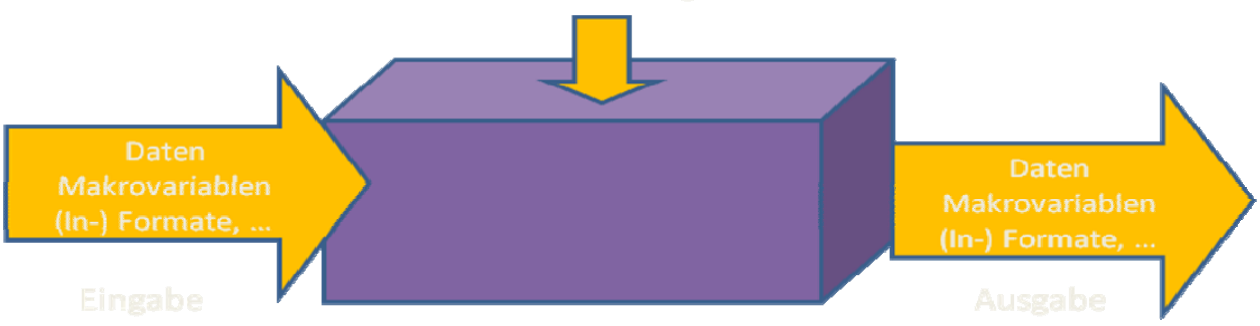

Abbildung 1: Blackbox-Test

2.3.2 Whitebox-Test

Bei dieser Testart ist die innere Struktur des Testobjekts bekannt, daher wird mit dieser Testart strukturorientiertes Testen durchgeführt. Die Testfälle werden gemäß des Kontrollflusses erstellt. Testziel hier ist die Coverage des Programms durch die Testdaten. Zusätzliche Implementierungen werden entdeckt. Da der Programmcode beim Whitebox-Test bekannt ist, gehört auch das Code-Review zu dieser Testart.

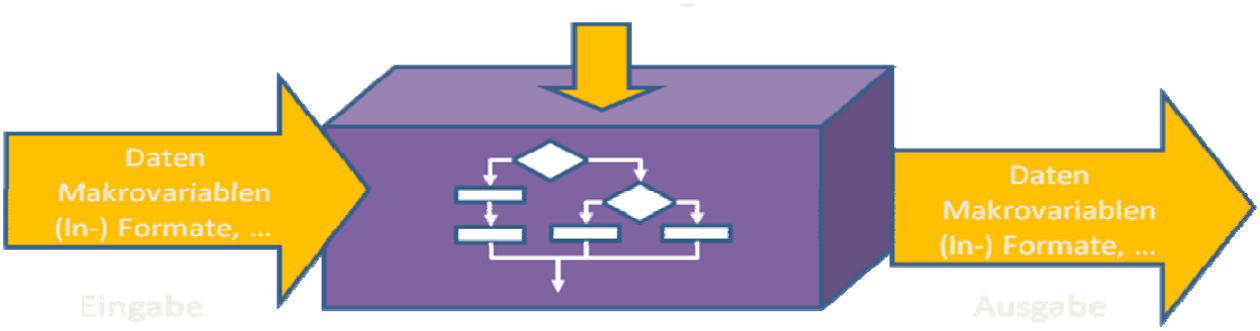

Abbildung 2: Whitebox-Test

2.4 Testphasen

2.4.1 Modultest

Der Modultest wird auch häufig Unittest oder Komponententest genannt. Testobjekte in dieser Testphase sind einzelne Makros und einzelne Programme. Der Modultest kann teilweise bereits während der Realisierungsphase durch den Entwickler durchgeführt werden. Der komplette Modultest sollte aber in einer getrennten Testphase durchgeführt werden. Ausgewählte Methoden des Modultests sind im Folgenden erläutert.

2.4.1.1 Code-Review

Beim Code-Review liegt der Schwerpunkt auf der Einhaltung der Entwicklerrichtlinien, insbesondere der Übersichtlichkeit, Wartbarkeit und Modularität des Testobjekts.

2.4.1.2 Testfälle

Mittels synthetischer Testdaten kann sowohl die fachliche Funktionalität als auch eine Überdeckung gezeigt werden. Bei der Erstellung der fachlichen Testdaten werden Äquivalenzklassen gebildet, aus deren korrekter Verarbeitung man auf die korrekte Verarbeitung ähnlicher Daten schließt. Werden die Testdaten anhand der Grenzen der Bedingungsanweisungen festgelegt, spricht man von Grenzwertanalyse. Unterstützend zu diesen beiden Teststrategien kann das Error-Guessing eingesetzt werden. Aus der Erfahrung mit bereits durchgeführten Projekten und dort vorkommenden Softwarefehlern werden solche Konstellationen konstruiert. Insgesamt sollte vor der Testdurchführung das erwartete Ergebnis des Tests festgelegt und dokumentiert werden, um anschließend leichter das tatsächliche Ergebnis des Tests beurteilen zu können.

2.4.1.3 Testdurchführung

Die Testdaten werden innerhalb eines Testtreibers erstellt, der das zu testende Objekt aufruft. Beim Coverage-Test wird nicht das zu testende Objekt selbst aufgerufen, sondern eine instrumentierte Version mit gleicher Funktionalität, aber erweitert um die Möglichkeit, anschließend die Überdeckungsquote zu bestimmen.

2.4.1.4 Testauswertung und Dokumentation

Entsprechend der zu Beginn festgelegten Testmetriken und der definierten Testfälle werden die Testergebnisse analysiert. Die gewählte Testmetrik ist dafür entscheidend, wie der Test zu bewerten ist. Die durchgeführten Test und Analyseergebnisse werden zur Nachvollziehbarkeit dokumentiert, meist werden auch alle notwendigen Dateien, die zu einer Wiederholung des Tests notwendig sind, archiviert.

2.4.2 Integrationstest

Schwerpunkt des Integrationstests sind die internen Schnittstellen zwischen den einzelnen Programmen und die fachliche Korrektheit des Softwareprodukts. Meist werden zum reinen Nachweis der technischen Funktionalität synthetische Testdaten, zum Nachweis der fachlichen Funktionalität echte Daten verwendet.

Die Erstellung der Testfälle, die Testdurchführung, die Testauswertung und Dokumentation ist analog zum Vorgehen beim Modultest. Aufgrund der meist hohen Komplexität des Softwareprodukts werden nicht alle möglichen Fälle untersucht, sondern nur diejenigen, von denen man erwartet, dass das Verhältnis zwischen Aufwand und Finden der Fehler nicht zu hoch ist.

2.4.3 Systemtests, Gesamtintegrationstest

Schwerpunkt des Systemtests, auch Gesamtintegrationstest oder end-2-end-Test genannt, sind die Schnittstellen zwischen den verschiedenen Softwareprodukten und deren fachlich korrektes Zusammenspiel. Meist werden echte Daten verwendet und die Systemumgebung sollte möglichst ähnlich zu der Systemumgebung der Produktion sein.

3 Fazit

Systematisches Testen erhöht die Wahrscheinlichkeit, dass die erstellte Software fachlich und technisch korrekt funktioniert und den Erwartungen entspricht. Definieren Sie vor Beginn der Tests Ihre Teststrategie und Testmetriken, nur so kann sichergestellt werden, dass Sie Ihre Testziele und damit ein qualitativ hochwertiges Softwareprodukt erstellen. Testen bedeutet nicht, die Richtigkeit des Programms zu zeigen, sondern es bedeutet, Fehler zu finden. Testen ist somit eher destruktiv, im Gegensatz zur Softwareentwicklung, die eher konstruktiv ist. Aufgrund dieser grundsätzlich anderen Ausrichtung sollten die Programme von Personen getestet werden, die nicht bei der Softwareerstellung beteiligt sind. Auch wenn man, wie E.W. Dijkstra formuliert hat, „Fehler nur

aufzeigen kann, niemals deren Abwesenheit“, erreichen Sie durch das systematische Testen eine höhere Qualität, so dass die von Ihnen erstellten Softwareprodukte nicht erst beim Kunden reifen müssen.