Tests in der Software-Entwicklung

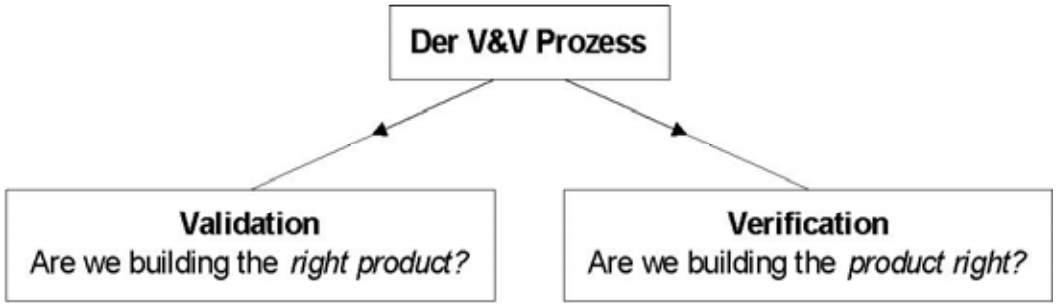
Die Qualitätsansprüche an Software steigen laufend – mit Recht, denn technologische Weiterentwicklungen haben nur dann einen Sinn, wenn sie Softwareprodukte zuverlässiger machen und ihre Bedienung vereinfachen. Diesen Standpunkt haben in den letzten Jahren auch die Anwender entwickelt. In den 80er und 90er Jahren wurde die Bedienung von Software oft noch als Geheimwissenschaft gehandelt. Über Programmfehler wurde leichter hinweggesehen, Hauptsache das Produkt erfüllte seine Funktion zumindest im Wesentlichen.

Im neuen Jahrtausend gelten diese Regeln nicht mehr. User sind zu Recht anspruchsvoller geworden. Sie erwarten sich zuverlässige Produkte, die stets (und ohne trickreiche Workarounds) ihre Aufgabe erfüllen. Das heißt, dass auch in der Softwareentwicklung andere Maßstäbe angelegt werden müssen. Dem Testen von Software kommt eine neue und zentrale Bedeutung zu.

Wir beschreiben im Folgenden, worum es beim Thema „Testen“ in der Software-Entwicklung geht. Wir geben eine Übersicht der wesentlichen Test-Bereiche, Test-Verfahren und Test-Tools und diskutieren ihre Vor- und Nachteile. Wenn Sie das Material gelesen haben, werden Sie einen Überblick haben, was beim Aufbau von Testumgebungen und der Auswahl von Test-Tools zu beachten ist.

1. Die Basics

In der Literatur wird alles, was mit Testen zu tun hat, unter dem Begriff "*Verification and Validation*" (V&V) zusammengefasst. Dabei ist folgende Unterscheidung zu treffen: *Validation* stellt die Frage, ob mit dem Produkt die Erwartungen des Auftraggebers erfüllt werden. *Verification* stellt die Frage, ob mit dem Produkt die Spezifikationen erfüllt werden.



In seiner Gesamtheit hat der V&V-Prozess zwei Ziele:

1. Auffinden von Defekten in einem System
2. Beurteilung eines Systems hinsichtlich Einsetzbarkeit in einer real-life Umgebung

2. Der Testprozess

Zumeist bestehen die zu testenden Systeme aus Subsystemen, die ihrerseits wieder aus einer Reihe von Modulen und Komponenten bestehen, welche wieder aus unzähligen Funktionen und Prozeduren bestehen. Daher sollte man ein Programm niemals als ein gesamtes Stück betrachten und testen. Das Testen sollte vielmehr im Abstimmung mit der Systemimplementierung erfolgen und die Aspekte der technischen Programmarchitektur in Betracht ziehen.

Daher gliedern sich Tests in folgende Subgruppen:

Komponenten-Tests beschäftigen sich mit dem Testen kleiner Teile (Units), die ihrerseits Module bilden:

- *Unit-Tests*: Hier werden kleinste Komponenten als stand-alone Teile betrachtet und auf korrekte Funktionalität getestet.
- *Modul-Tests*: Module sind die nächst größeren Einheiten, die zu testen sind. Mehrere voneinander abhängige Units bilden ein Modul, welches im Modul-Test auf Korrektheit getestet wird.

Integrations-Tests dienen dazu, größere Teile eines Systems auf Herz und Nieren zu testen. Sie gliedern sich in Subsystem- und System-Tests:

- *Subsystem-Tests:* Hier testet man die korrekte Zusammenstellung von Modulen zu Subsystemen, vor allem hinsichtlich der Interfaces zu anderen Subsystemen.
- *System-Test:* Man versucht in dieser Phase jene Fehler zu finden, die aufgrund schlechter Interaktion von Subsystemen passieren. Hier testet man weiters, ob das Produkt den funktionalen und nicht-funktionalen Anforderungen entspricht.

3. Testverfahren

Im Folgenden beschreiben wir die beiden wesentlichen Testverfahren, Black-Box beziehungsweise White-Box Tests. Vorweg sei angemerkt, dass ein gut gewählter Mix aus Black-Box und White-Box Tests in der Praxis die am besten geeignete Lösung darstellt.

Black-Box Tests

Bei einem Black-Box Test kennt der Tester nur die möglichen Input-Werte sowie die erwarteten Output-Werte für ein zu testendes Modul oder System, kurz gesagt: die Anforderungen. Der technische Aufbau des Moduls ist gänzlich unbekannt und jegliches Wissen über die Struktur des Moduls wird für die Tests nicht verwendet.

Vorteile von Black-Box Tests:
- kostengünstig
- sind fokussiert auf die Anforderungen, was bei White-Box Tests oft vergessen wird
- flexibler bezüglich Implementierungsänderungen

Nachteile von Black-Box Tests:
- mitunter werden nicht alle Programmfehler gefunden

White-Box Tests

Diese Tests erfordern tiefgehendes Wissen über den technischen Aufbau des zu testenden Moduls/Systems. Die Programmstruktur des Testobjekts wird genauestens analysiert und getestet.

Vorteile von White-Box Tests:
- es werden auch Fehler in "versteckten" Teilen des Programms gefunden
- der Tester wird gezwungen, sich mit der technischen Struktur des Moduls auseinanderzusetzen

Nachteile von White-Box Tests:
- kostenintensiv
- wenn Teile der Funktionalität fehlen, könnte das bei White-Box Tests übersehen werden.

4. Testtools

In diesem Abschnitt beschreiben wir die unterschiedlichen Werkzeuge, die für das Testen von Software eingesetzt werden. Im Wesentlichen lassen sich diese Tools in zwei Kategorien einteilen: Jene zur *Planung und Organisation* der Tests und solche, welche die *Durchführung* der Tests selbst unterstützen.

Tools zur Unterstützung der Testplanung und -organisation

Mit der wachsenden Bedeutung des Testens für die Softwareentwicklung ist eine Reihe von Tools entstanden, welche die Tester bei der Testplanung und bei der Testorganisation unterstützen. Ein wesentlicher Grund dafür ist, dass im gesamten Bereich des Testens die Kommunikation zwischen den beteiligten Akteuren ein ausgesprochen wichtiger Punkt ist. Daher ist es unerlässlich, die Kommunikation zwischen den Testern, den Entwicklern und den übergeordneten Stellen zu standardisieren. Hier die wichtigsten Anforderungen an ein Tool für Testplanung und –organisation:

- *Erfassung der Requirements*
  Ein Testtool sollte es ermöglichen, die Requirements an das zu testende System zu erfassen und zu verwalten. Im einfachsten Fall reicht eine Aufzählung der einzelnen Anforderungen mit der Möglichkeit, Text- oder sonstige Dokumente anzuhängen. Den Requirements werden Testfälle zugeordnet, wodurch man in Folge die Testabdeckung der einzelnen Systemteile überprüfen kann.

- *Erfassung und Organisation der Testfälle*
  Dies beinhaltet die Möglichkeit, Testfälle zu erstellen und zu verwalten. Ein Testfall ist ein Szenario aus dem "Leben" eines Systems. Dieses Szenario wird für den Tester in einzelne Testschritte zerlegt, damit es am Testsystem durchgespielt werden kann. Jeder Testfall besitzt ein erwartetes Ergebnis, bei dessen Erreichen der Testfall auf "positiv" gesetzt werden kann.

- *Organisatorische Unterstützung bei der Durchführung von Testfällen*
  Testfälle werden im Rahmen der Testdurchführung zu Testsets zusammengefasst. Ein Testplanungs und -organisationstool sollte die Möglichkeit geben, Testsets zu kreieren und diese zu verwalten.

- *Defektverwaltung*
  Führt ein Testfall zu einem negativen Ergebnis, so muss ein Defekt erfasst werden. Der Defektverwaltung sollte in einem Testorganisationstool großes Augenmerk zukommen. Die Defekterfassung sollte in jener Art und Weise standardisiert werden, sodass jeder Involvierte den Defekt versteht.

- *Schnittstellen zu Testdurchführungstools*

Häufig besitzen Testplanungs und -organisationstools Schnittstellen zu bestimmten Testdurchführungstools. Hauptsächlich werden Anbindungen zu Automatisierungs- und Simulationstools angeboten, indem es ermöglicht wird, diese Art von Testfällen gemeinsam mit den anderen zu verwalten und durchzuführen.

Tools zur Unterstützung der Testausführung

Tools dieser Art gibt es seit den Frühzeiten professioneller Software-Entwicklung. Heute sind eine Reihe von Testdurchführungstools im Einsatz, die alle unterschiedliche Bereiche abdecken:

- *Profiler*
  Mit einem Profiler kann man das zu testende Programm durchlaufen lassen und Laufzeitanalysen erstellen. Diese Analysen ermöglichen es in Folge, performancebedingte Schwachpunkte aufzuzeigen und zu korrigieren.

- *Debugger*
  Ein Debugger ist das klassische White-Box Unit-Testtool. Mit Hilfe eines Debuggers befindet man sich während des Ablaufes eines Codesegmentes direkt beim Programmcode und kann jede Programmzeile einzeln ablaufen lassen und analysieren.

- *Simulationstools*
  Mit Hilfe von Simulationstools lassen sich bestimmte Umgebungsbedingungen für ein Programm simulieren. Es könnte zum Beispiel zur Simulation eines Lastentests eine bestimmte Anzahl von Transaktionen/Sekunde von außen generiert werden, um die Performance des Systems unter Last zu testen. Ebenfalls fallen unter diese Kategorie jene Tools, die zum Zwecke eines Stresstests bestimmte Systemressourcen bis an die Grenze auslasten, um das Verhalten des Programms in Extremsituationen zu beobachten.

- *Testdatengeneratoren*
  Um repräsentative Tests durchzuführen ist es oft notwendig, eine Reihe von Testdaten zur Verfügung zu haben. Daher kann es erforderlich sein, für die Testdatengenerierung ein geeignetes Tool zu verwenden.

- *Automatisierungstools*
  In letzter Zeit gewinnen diese Art von Tools immer mehr an Bedeutung. Mit einem Automatisierungstool kann ein Tester bestimmte Anwendungsszenarien ohne manuellen Eingriff ablaufen lassen. Diese Testmethode ist vor allem dann zu erwägen, wenn das zu testende System eine große Anzahl von heterogenen Input-Datenmengen zu verarbeiten hat.

5. Menschliche Aspekte des Testens

Abschließend sollte noch auf einen Punkt eingegangen werden, der in der Literatur weitgehend unerwähnt bleibt, nämlich die menschlichen Aspekte des Testens.

Ein Tester hat eine sozial unangenehme Aufgabe: *Er zeigt Fehler seiner Kollegen auf.* Natürlich wird niemand gerne auf seine Fehler aufmerksam gemacht, daher besteht oft zwischen dem Tester und seinen Kollegen ein gewisses Spannungsfeld.

Verbesserungsansätze für Tester: Tester sollten sich einen gewissen Grad an Sensibilität aneignen, um die „entlarvten" Kollegen nicht allzu sehr zu brüskieren. Wichtigste Grundregel: *Fehlermeldungen sollten immer sachlich gehalten sein und niemals persönliche Elemente enthalten.*

Verbesserungsansätze für Entwickler: Entwicklern müssen sich klar machen, dass es menschlich ist, Fehler zu machen und dass jeder gelegentlich Fehler produziert. Wichtig ist, dass falls ein Fehler aufgedeckt wurde, dieser sorgfältig und rasch verbessert wird und in Zukunft nicht erneut gemacht wird. Wichtigste Grundregel: *Den Fokus weg von der Schuldfrage und hin auf die Korrektur des Fehlers lenken.*

Ergänzend zu diesem Text steht Ihnen auf der InfraSoft Website ein Glossar zur Verfügung, in dem Sie die meisten der hier verwendeten Begriffe finden. Über das aktuelle Angebot an weiteren, kostenlosen Fachbeiträgen zur Softwareentwicklung informieren Sie sich bitte unter www.infrasoft.at/service.

Mag. Andreas Fandl
Wien, im Oktober 2002

Der Autor ist Mitarbeiter der InfraSoft, einem Unternehmen, das auf komplexe Softwareentwicklungen spezialisiert ist. Die Experten der InfraSoft haben langjährige Erfahrungen in der Entwicklung und verfügen über fundierte Kenntnisse in Design, Analyse, Realisierung, Test und Projektmanagement.
Für **individuelle Beratungen** zur Entwicklung von Softwarelösungen und die Bereitstellung von **Realisierungsteams** wenden Sie sich bitte an info@infrasoft.at.