Testverfahren für Software *

Lee Sprague **

Abstract

Obwohl perfekte Software nicht existiert, kann die Anwendung gründlicher Testverfahren doch die Anzahl möglicher Fehler erheblich vermindern. Je eher Testverfahren eingesetzt werden, desto größer ist die Einsparung bei den Entwicklungskosten.

Unabhängige Software-Testinstitute arbeiten mit Standardisierungsorganisationen zusammen, um Teststandards zu definieren und zu etablieren. Solche Laboratorien sorgen für objektives, präzises und wirtschaftliches Testen, indem sie detaillierte Testpläne und Testprozeduren verwenden, die sicherstellen, daß alle Items getestet werden und daß die Tests wiederholbar sind. Testpläne identifizieren Testniveaus, Testtypen, Testmethoden und Testprozeduren.

Die Testniveaus sind „Einheit", „System" und „Integration". Testtypen sind: Funktionstest, Destruktionstest, Regressionstest und Dokumentationstest. Testmethoden sind: Prozedurales Testen, Strukturtesten, „ad hoc"-Testen, Vergleichstesten und Kompatibilitätstesten. Software-Testprozeduren umfassen die manuelle Eingabe, die halbautomatische Eingabe oder die automatische Eingabe und gegebenenfalls die optische Prüfung der Ausgabeergebnisse.

Einleitung

Keine Software ist fehlerfrei, und sie verhält sich bei der ersten Verwendung typischerweise nicht wie antizipiert. Auch kommt es vor, daß Programme jahrelang ohne Beanstandung laufen und doch verborgene Fehler enthalten. Die Fehlerwahrscheinlichkeit nimmt rapide zu, wenn die Komplexität von Software steigt oder Software mit anderer Software integriert wird. Obwohl es keine fehlerlose Software gibt, verringert der Einsatz von gründlichen Testverfahren doch die Anzahl möglicher Fehler. Außerdem sinken die Entwicklungskosten um so stärker, je eher diese Testprozeduren eingeführt werden.

Warum kann Software nicht perfekt sein?

Software-Fehler treten immer dann zutage, wenn Software nicht wie antizipiert funktioniert. Weil einige Computerprogramme viele tausend Zeilen lang sind, übersteigt es häufig die menschlichen Fähigkeiten, etwas derart Großes und Komplexes fehlerfrei zu konstruieren. Sogar vermeintlich kleine Fehler können katastrophale Folgen haben, wie die folgenden Beispiele zeigen:

1. Ein falsch plaziertes Zeichen auf einer einzigen Lochkarte führte in einer Stadt in Neu-England zu der irrigen Annahme, daß das Steueraufkommen sieben Millionen Dollar höher als in Wirklichkeit sei. Da die Steuerrate zu niedrig angesetzt wurde, ging der Stadt das Geld aus.

2. Fehlberechnungen, die in die Computer des „Bureau of Reclamation" eingespeist wurden, hatten zur Folge, daß zu viel Wasser hinter den Dämmen zurückgehalten wurde, was wiederum Überschwemmungen längs des Colorado-Flusses verursachte.

3. Der Aktienindex von Vancouver verlor aufgrund eines Computerfehlers über ein Jahr lang täglich einen Punkt. Als der Fehler gefunden wurde, hatte der Index bereits insgesamt 574 Punkte verloren.

4. Ein Flugzeug der „United Airlines" verlor bei einem Flug nach Denver an Geschwindigkeit und Höhe. Vermutlich lag das an einem Bordcomputer, der nicht auf fallende Außentemperaturen reagierte, so daß die Motoren erst vereisten und dann überhitzt wurden.

5. Während des Falklandkrieges sank der britische Zerstörer „Sheffield", weil das Abwehrsystem des Schiffes eine anfliegende Exocet-Rakete als Waffe eines Verbündeten einstufte und deshalb nicht angriff.

6. Der Jungfernflug des amerikanischen Space Shuttle wurde durch einen Fehler in einer der 500.000 Programmzeilen verzögert.

Alle diese Fälle hatten beträchtliche finanzielle Auswirkungen. William Goss, Präsident des „International Bureau of Software Test, Inc." hatte mit Computersystemen zu tun, die jahrelang erfolgreich liefen, bevor ein Fehler zu Tage trat. So führte beispielsweise eine bestimmte Zahlenfolge, die so vorher noch nicht aufgetreten war, nach Jahren zum Versagen einer Gleichung.

Alan Borning, ein Computerwissenschaftler an der Universität Washington, schätzt, daß ein „Star Wars"-Verteidigungssystem ungefähr zehn Millionen Zeilen an Software-Code erfordert. Er sagte: „Man kann es nicht einmal adäquat testen. Schließlich kann man nicht zu den Russen sagen 'Schießt ein paar Raketen ab, wir wollen unser Programm testen.' Es muß beim ersten Mal fehlerlos arbeiten, ohne vorher irgendwie in realistischer Weise getestet worden zu sein." (1)

* Dieser Artikel wurde zuerst in den Akten der Nationalen Computerkonferenz von 1985 gedruckt. Die Übersetzung erscheint hier mit freundlicher Genehmigung des „International Bureau of Software Test, Inc.", 536 Weddell Drive, Suite 7, Sunnyvale, CA 94089, USA.

** Lee Sprague ist Vizepräsident des „International Bureau of Software Test".

In „Software Testing Techniques" hat Boris Beizer darauf hingewiesen, daß ein aus zehn Buchstaben bestehender Eingabestring 2 hoch 80 mögliche Eingabeströme mit den jeweils entspechenden Ausgaben erlaubt, und dazu angemerkt, daß ein vollständiges funktionales Testen in diesem Sinne offensichtlich höchst unpraktikabel ist. Bei einer Mikrosekunde pro Einzeltest würde die Gesamttestzeit ungefähr viermal so lang sein wie das geschätzte Alter des Universums (2). Das zeigt, wie schwer Software es hat, perfekt zu sein.

Was ist der Stand von Testverfahren für Software?

Bis vor kurzem waren Testprozeduren für Software eher unorganisierte Verfahren, die nur wenige Standards und kaum Konsistenz besaßen. Der Großteil des professionellen Testens fand innerhalb weniger großer Organisationen statt, in denen spezialisierte Abteilungen die vom Unternehmen entwickelten Programme testeten. Unabhängige Autoren und kleine Softwareentwicklungsgesellschaften waren hinsichtlich des Softwaretests von den im Hause beschäftigten Programmierern oder von Freunden abhängig. Es ist jedoch für Softwareentwickler extrem schwer, ihre eigenen Programme effektiv zu testen, weil sie selber keine Fehler machen, wenn sie Daten in ihre eigenen Programme eingeben. Es bereitet ihnen Schwierigkeiten, sich vorzustellen, welche Fragen und welche Verwirrung sich bei dem ungeübten Anwender einstellen können. Freunde der Softwareentwickler besitzen ebenfalls nicht die notwendige Unparteilichkeit, die notwendig ist, um in den Programmen ihrer Freunde Fehler zu provozieren beziehungsweise zu entdecken. Aus den genannten Gründen ist eine Reihe von neuen Softwareprogrammen nicht gründlich und objektiv ausgetestet worden.

Unabhängige Laboratorien für Software-Tests können für objektives, schnelles, gründliches und wirtschaftliches Testen sorgen, weil ihr vordringliches Interesse gerade darin besteht, Softwaremängel so früh wie möglich zu entdecken. Frühes Intervenieren aber bewirkt geringere Kosten beim Testen und bei der Überarbeitung.

Warum werden Pläne und Prozeduren für den Test von Software verwendet?

Pläne und Prozeduren für den Test von Software garantieren ein methodisches Provozieren von Softwarefehlern. Sie sorgen ebenfalls dafür, daß alle Items getestet und verifiziert werden und daß die Tests wiederholbar sind. Ohne Testprozeduren müssen sich Testingenieure auf ihre Intuition in einem gegebenen Moment verlassen. Mit Testplänen und Testprozeduren können Testingenieure für die Rekonstruktion der Tests die genaue Abfolge oder die Tastendrücke wiederholen.

Was gehört zu einem Testplan für Software?

Ein Testplan definiert den Projektumfang und beschreibt die Ziele und Methoden des Tests. Er legt die Eigenschaften des Produkts dar und beinhaltet einen Zeitplan mit zeitlichem Rahmen und Festlegungen dazu, wann die einzelnen Testphasen abzuschließen sind. Außerdem benennt der Testplan die Testniveaus, Testtypen und Testmethoden.

Was sind Testniveaus?

Unter Testniveau versteht man den Testgrad, der für die einzelnen Produktmodule zu verwenden ist. Die drei Elementarniveaus sind „Einheit", „System" und „Integration"; für jedes Projekt legt der Testzweck eines oder mehrere dieser Testniveaus fest.

Niveau „Einheit"

Dieses Niveau erfordert eine Isolation jeder Einheit von allen anderen Einheiten, sowie die Identifizierung und Kontrolle der Übergänge zu allen anderen Einheiten.

Niveau „System"

Alle Eigenschaften und Funktionen innerhalb eines einzelnen Systems werden in ihrem Zusammenwirken getestet. Die Interaktion zwischen den Einheiten des Systems und den Verbindungen zwischen den Einheiten wird getestet.

Niveau „Integration"

Hier werden alle Eigenschaften und Funktionen von mehr als einem System in ihrem Zusammenwirken getestet. Das kann das Testen der beabsichtigte Systemkonfiguration im Ganzen beinhalten und schließt Software, Firmware und/oder Peripherie ein. Sind mehrere Anwendungen möglich, so beinhaltet das Niveau „Integration" ebenfalls den Test aller Eigenschaften und Funktionen in ihrer Beziehung zueinander. Wenn also zum Beispiel Textverarbeitungs-, Tabellenkalkulations-(spreadsheet), Rechtschreibprüfungs(spelling checker) sowie Zeitplanungsprogramme integriert sind, so müssen sie alle in ihrem Zusammenwirken getestet werden.

Was sind die vier Haupttestarten?

Die vier Haupttestarten sind der Funktionstestest, der Destruktionstest, der Regressionstest und den Dokumentationstest.

Funktionstest

Testfolgen für den Funktionstest orientieren sich am normalen Gebrauch, der vom potentiellen Anwender

zu erwarten ist. Zum Funktionstest gehört beispielsweise die Überprüfung von Grenzwerten, Paßworten, Menüs, Bildschirmdarstellungen, Fehlermeldungen und Prompts. Dieser Test kann auch die Klarheit von Meldungen und Aufbau prüfen. Er kann des weiteren untersuchen, ob die Antwortzeiten akzeptabel sind, ob das Benutzer-Interface in Ordnung ist und ob alle Konfigurationen arbeiten. Zusätzlich könnte er verschiedene Funktionskombinationen testen. Schließlich kommt noch ein Test der Akkuratheit von Fehlerbehandlungsroutinen und der Verwendung von Programmabschlußsequenzen in Betracht.

Destruktionstest

Der Destruktionstest kalkuliert unnormale Anwendungsbedingungen bei der Nutzung durch den potentiellen Anwender ein. Bereiche für den Destruktionstest sind: Werte außerhalb vorgesehener Grenzen, unzureichender Umfang von Hauptspeicher und/oder externem Speicher, unzureichende Datei-Struktur und Überschreiten des maximal möglichen Zahlenbereichs. Schließlich prüft der Destruktionstest noch, wie simulierte Hardwarefehler abgefangen werden und auf welche Weise ungültige Eingaben (oder Kombinationen davon) zurückgewiesen werden.

Regressionstest

Der Regressionstest wiederholt den Funktions- bzw. Destruktionstest um zu überprüfen, ob festgestellte Diskrepanzen korrigiert worden sind und ob nicht neue Diskrepanzen als eine Folge der Programmänderungen oder Systemkorrekturen aufgetreten sind.

Dokumentationstest

Der Dokumentationstest verifiziert, daß die Dokumentation und das System übereinstimmen und daß die Dokumentation gut organisiert, leicht zu verstehen und technisch und grammatisch in Ordnung ist. Insbesondere überprüft der Dokumentationstest, ob das Handbuch Fragen des Benutzers beantwortet und ob es klar und konsistent ist.

Was sind Testmethoden?

Zu den Software-Testmethoden gehören: Prozedurales Testen, Strukturtesten, „ad hoc"-Testen, Vergleichstesten, Kompatibilitätstesten.

Prozedurales Testen

Prozedurales Testen benutzt strikte Richtlinien für jeden Testpfad, so daß jede in der Dokumentation beschriebene Eigenschaft und Funktion getestet wird. Eine Testprozedur ist hier eine spezifische Abfolge, die (Tastendruck für Tastendruck) das für den Testfall definierte Resultat erreicht. Testprozeduren stellen wiederverwendbare, konsistente, Schritt für Schritt festgelegte Richtlinien für künftiges Testen zur Verfügung.

Prozedurales Testen hat den Vorteil, daß eine detaillierte Dokumentation für den Test aller Bereiche des Produkts zur Verfügung steht, so daß Tester Problemzonen rekonstruieren können. Die Nachteile des prozeduralen Testens liegen darin, daß es zeitaufwendig ist, daß es kein intuitives Testen erlaubt und daß es — werden beim Testdesign Bereiche übersehen — in diesen Bereichen keine Diskrepanzen aufdeckt.

Strukturtesten

Testingenieure benutzen das Strukturtesten, um sich den Weg durch das Testen miteinander verknüpfter Teile des Systems zu bahnen. Alle Eigenschaften, die in der Strukturbeschreibung aufgelistet sind, werden getestet, aber nicht notwendigerweise in der Reihenfolge, in der sie aufgeführt sind. Strukturtesten ist teilweise eine intuitive Testmethode, weil der Tester logischen Pfaden folgt, wo immer sie auch hinführen mögen. Die Tester legen Testprotokolle an, die einen Nachweis für den Verlauf und die Logik des Testens darstellen.

„ad hoc"-Testen

Besonders geschulte Testingenieure benutzen für bestimmte Bereiche das „ad hoc"-Testen. Es werden keine anderen Dokumentationen als Berichte über Software-Diskrepanzen erstellt. Der Testingenieur richtet sich nach nichts anderem als seiner Intuition und der zur Verfügung gestellten Dokumentation.

Vergleichstesten

Vergleichstesten bestimmt die Ähnlichkeiten und Unterschiede zwischen zwei oder mehr Systemen. Es kann von dem Nachweiskapitel der Begleitdokumentation ausgehen, dann eine Eigenschaftsvergleichsmatrix aufbauen und schließlich jeden Befehl zuerst auf dem einen und dann auf dem anderen System testen. Darauf folgt ein Vergleich der Resultate und eine Dokumentation der Unterschiede.

Wenn die zu testenden Systeme kompatibel sind, besteht eine Alternative darin, einen automatisierten Test gleichzeitig auf beiden Systemen laufen zu lassen, die Resultate zu vergleichen und die Unterschiede zu dokumentieren. Wenn externe Hardware wie etwa Drucker beteiligt ist, kann Vergleichstesten darin bestehen, daß man ein Hauptdokument erstellt, das alle denkbaren Eigenschaften und Funktionen ausnutzt, und dieses Dokument dann auf jedem Drucker ausgibt. Die Resultate werden verglichen und die Unterschiede dokumentiert.

Kompatibilitätstesten

Kompatibilitätstesten bestimmt die Fähigkeit von zwei oder mehr Systemen, in austauschbarer Weise zu

funktionieren, und identifiziert funktionale Unterschiede. Hier interessierende Bereiche sind u.a. Tastatureingabe, Bildschirmausgabe, reproduzierbare Diskrepanzen, Betriebssystem-Funktionen, Eingabe/Ausgabe-Funktionen und Diskettenaustausch.

Was sind Software-Standards?

Software-Standards können so formal sein wie ein festgelegter Industriestandard (wie z.B. der „ANSI FORTRAN 77 Standard") oder die Informationsspezifikationen darstellen, unter denen das Produkt entworfen wurde. Das „International Bureau of Software Test" unterstützt ACM und IEEE bei der Erstellung und Definition von Softwaretest-Standards.

Was sind Testprozeduren?

Testprozeduren sind in detaillierten Dokumenten festgehalten, die spezifische Schritte zur Erreichung der jeweiligen Testziele beschreiben. Eingeschlossen sind Kriterien für Annahme oder Verwerfung, wie auch eine Beschreibung der Konfiguration, die für die Durchführung des Tests nötig ist. Obwohl die Komplexität der Prozeduren mit dem Testniveau variiert, ist die Form der Prozeduren doch ähnlich. Softwaretest-Prozeduren verlangen von dem Testpersonal, daß das gesamte Software-Produkt auf Inhalt, Präsentation, Organisation und Einhaltung der Dokumentation hin durchgesehen wird. Diese Prozeduren verlangen manuelle Eingaben und optische Auswertungen (oder eine Kombination von beidem). Der Ausführungstest kann aus manueller Eingabe, halbautomatischer Eingabe und/oder automatischer Eingabe bestehen.

Was sind Vor- und Nachteile einzelner Testprozeduren?

Manuelle Eingabe

Testingenieure geben hier manuell Daten und Kontrollinformationen so ein, wie dies für die Testprozeduren schriftlich festgelegt ist. Der hauptsächliche Vorteil der manuellen Eingabe besteht in dem direkten Übergang von der Entwicklung der Prozedur zu deren Ausführung durch den Testingenieur im Rahmen des Tests. Die Hauptnachteile sind, daß erstens die manuelle Eingabe die Möglichkeit von Eingabefehlern mit sich bringt und daß zweitens während der gesamten Dauer des Tests und der Auswertung Personal benötigt wird. Dieser Personalbedarf beeinflußt die Testkosten, da die Personalinvestitionen bei jedem Testlauf in gleicher Höhe erneut anfallen.

Halbautomatische Eingabe

Die halbautomatische Eingabe besteht teils aus manuellen Prozeduren und teils aus der Ausführung von Softwaretest-Code. Hauptvorteile der halbautomatischen Testprozeduren sind erstens der eingeschränkte Bedarf für die Entwicklung von Testcode undzweitens die verringerten Anforderungen an Anwesenheit und Mitwirkung von Personal. Diese Vorteile sorgen für Kostenkontrolle. Der Hauptnachteil der halbautomatischen Durchführung ist darin zu sehen, daß die manuelle Dateneingabe zu Eingabefehlern führen kann.

Automatische Eingabe

Der Testingenieur führt den Testcode aus (und zwar einschließlich der automatischen Verifizierung der Testresultate) und wertet die Ergebnisse aus. Die Hauptvorteile der automatischen Eingabe sind erstens die Möglichkeit, den Test jederzeit in genau der gleichen Weise zu wiederholen, zweitens die während der Testdurchführung erheblich reduzierten Personalanforderungen (der Tester braucht den Test nur zu starten und kann die Resultate nach dessen Beendigung auswerten) und drittens das Fehlen einer menschlichen Irrtumsmöglichkeit. Der Hauptnachteil der automatischen Eingabe liegt in den Investitionskosten für den Testcode und darin, daß Testinstrumente vor Durchführung des Tests entwickelt werden müssen. Man muß jedoch sehen, daß diese hohen Kosten sich mit der Anzahl der Testwiederholungen amortisieren.

Wann sollten Testprozeduren eingeführt werden?

Die größte Einzelkomponente der Softwarekosten ist das Testen mit dem Ziel, Softwarefehler aufzudekken. Testkosten können mehr als 50% der Software-Entwicklungskosten betragen. Deswegen sollten Testprozeduren so früh wie möglich im Entwicklungszyklus eingeführt werden. Je eher Fehler gefunden werden, desto weniger kostenaufwendig ist es, sie zu korrigieren. Wenn Tester Design-Spezifikationen überprüfen, können sie die Aufmerksamkeit auf Problembereiche lenken und auf Mängel achten. Die Tester können außerdem Testpläne schon entwickeln, während die Programmierer noch programmieren, so daß das Testen sofort beginnen kann, wenn die Programmierung beendet ist.

Schlußfolgerungen

Obwohl es keine perfekte Software gibt, verringert die Anwendung gründlicher Testverfahren doch die Anzahl möglicher Fehler erheblich. Hinzu kommt, daß je früher Testprozeduren eingesetzt werden, je größer die daraus resultierenden Kostenersparnisse sind.

Nachweise

(1) Borning, A.: „The Last Bug Computer Scientists Fear", San Jose Mercury, 29. Oktober 1984, S. 1.

(2) Beizer, B.: Software Testing Techniques, New York 1983.