11 Testautomatisierung

Dieses Kapitel erklärt zuerst einige grundlegende Definitionen und Konzepte der Testautomatisierung, sowie einen Prozess zur effizienten Auswahl von Testwerkzeugen. Es liefert sodann einen Überblick über Arten von Testwerkzeugen und über am Markt erhältliche Testwerkzeugen. Da die automatisierte Durchführung von funktionalen Tests viel Verbesserungspotential in sich birgt, aber auch sehr problematisch sein kann, wird auf diese Art der Automatisierung näher eingegangen.

11.1 Definitionen und Hintergründe

Testautomatisierung ist die Verwendung eines Testwerkzeugs (oder Testtools) mit dem Ziel, das Testen zu unterstützen.

> *„Ein Testtool ist ein automatisiertes Hilfsmittel, das bei einer oder mehreren Testaktivitäten, beispielsweise Planung und Verwaltung, Spezifikation, Aufbau von Ausgangsdateien, Testdurchführung und Beurteilung, Unterstützung leistet.“ [Pol et al, 2000].*

Der Nachdruck liegt hierbei auf der Unterstützung. Ein Werkzeug ist nur dann ein Hilfsmittel, wenn sein Einsatz zu höherer Produktivität und Effizienz führt. Verwendung des Werkzeug per se darf nicht das Ziel sein.

Testwerkzeuge umfassen sowohl kommerziell erhältliche Werkzeuge als auch selbst geschriebene Programme für die automatische Durchführung von Testaktivitäten.

Vorteile	**Nachteile**
Eine umfangreiche Menge von Tests können ohne Kontrolle und automatisch[1] durchgeführt werden, z.B. über Nacht. Dies gilt besonders für Regressionstests.	Die meisten kommerziell erhältlichen Werkzeuge sind sehr teuer.
Automatisierung von sich wiederholenden und oft lästigen Testaktivitäten führt zu höherer Zuverlässigkeit dieser Aktivitäten und zu höherer Zufriedenheit im Testteam, was wiederum zu höherer Produktivität führt.	Viele Testwerkzeuge erfordern viel Aufwand für den erfolgreichen Einsatz, und sind daher nur profitabel für große Projekte, oder für große Firmen mit vielen Projekten.

[1] Allerdings existieren sehr wenige Werkzeuge zur Test-Durchführung, die eine Fortsetzung der Tests nach schweren Fehlern in der Applikation erlauben. Wenn solch ein Fehler während einer langen Phase ohne Überwachung stattfindet, kann der Test für eine lange Zeit stillstehen. Baut der Tester auf die erfolgreiche Testdurchführung, so kann das zu schwerwiegenden Verzögerungen im Testprozess führen.

Vorteile	Nachteile
Testwerkzeuge sorgen dafür, dass die Testdaten bei aufeinander folgenden Tests die gleichen sind, so dass Gewissheit über die Zuverlässigkeit der Ausgangssituation und der Daten besteht.	Die Verwendung von Testwerkzeugen erfordert einen strukturierten Testprozess und viel Disziplin. Unternehmen, welche diese Kriterien nicht erfüllen, riskieren, ihre Marktsituation zu verschlechtern, indem sie teuer Werkzeuge kaufen und dann nicht einsetzen.
Werkzeuge können Fehler finden, die manuell nur schwer zu erkennen sind, z.B. memory leaks.	
Das Erzeugen großer Mengen an Testdaten kann automatisch mit Hilfe eines Testwerkzeugs erfolgen. Die Eingabe von initialen Daten muss lediglich einmal erfolgen und nicht bei jedem Test erneut.	

Tabelle 11.1: Vor- und Nachteile des Gebrauchs von Testwerkzeugen [Pol et al, 2000]

Die Verwendung eines oder mehrerer Testwerkzeuge macht Sinn, wenn auf einen strukturierten Testprozess mit passenden Methoden aufgebaut werden kann. In einem kontrollierten Prozess können Werkzeuge eine wichtige Ergänzung darstellen, in einem unzureichend kontrollierten Prozess können sie jedoch kontraproduktiv sein. Automatisierung erfordert die Durchführbarkeit und eine gewisse Standardisierung der zu unterstützenden Aktivitäten. Da Automatisierung eine bestimmte standardisierte Arbeitsweise voraussetzt, kann sie dabei helfen, einen strukturierten Prozess einzuführen.

Testautomatisierung ist also sowohl mit Vor- als auch Nachteilen behaftet. Tabelle 11.1 führt einige der Wichtigsten an.

11.2 Auswahl von Testwerkzeugen

Testmanager müssen Testwerkzeuge mit Bedacht auswählen, besonders wenn Testwerkzeuge in Betracht gezogen werden, die einen erheblichen finanziellen Beschaffungsaufwand bedeuten oder der effiziente Einsatz dieser Werkzeuge mit großem Mehraufwand verbunden ist. Der Prozess der Selektion selbst ist eine sehr zeitaufwendige Arbeit. Kriterien für die Werkzeugauswahl müssen sorgfältig auf die Testanforderungen abgestimmt werden. Jedes Werkzeug muss gegen diese Kriterien getestet werden.

Es existieren bereits Sammlungen von Daten über verschiedene erhältliche Testwerkzeuge, z.B. [Graham et al, 1995]. Solche Arbeiten können bei der Auswahl sehr hilfreich sein; allerdings muss, besonders bei älteren Sammlungen, berücksichtigt werden, dass zwischen Veröffentlichung der Sammlung und Werkzeugauswahl bereits neue, bessere Werkzeuge herausgekommen sein könnten.

Abbildung 11.1 zeigt einen aus vier Schritten bestehenden Prozess für die erfolgreiche Auswahl von Testwerkzeugen[2] [Poston, 1992]:

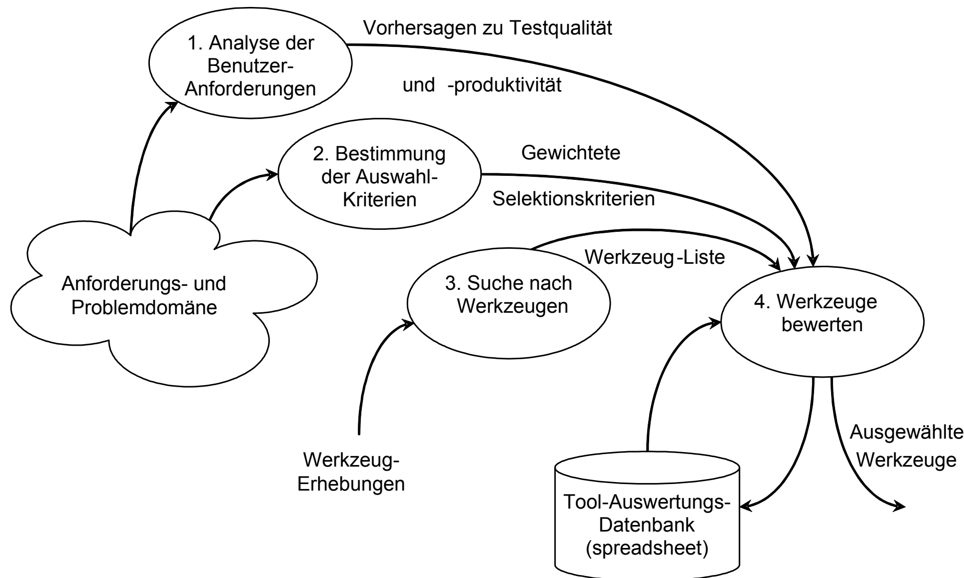

Abbildung 11.1: Prozess zur Auswahl von Testwerkzeugen nach Poston [Poston, 1992]

1. Der Auswertende muss die Anforderungen der Benutzer identifizieren und quantifizieren. Das bedeutet, dass er die Art des benötigten Werkzeugs ermitteln muss, z.B. unter Verwendung einer Tabelle wie Tabelle 11.2, und dass er Daten zu den Kosten und zur Produktivität schätzen muss, z.B. die Testkosten pro Phase oder die gewünschte Testüberdeckung
2. Als nächstes ermittelt der Auswertende Kriterien zur Werkzeugauswahl, z.B. Preis oder Anforderungen an die Funktionalität, und gewichtet sie. Keine zwei Kriterien dürfen dasselbe Gewicht bekommen.
3. Der Auswertende sucht nach vorhandenen Werkzeugen mittels Erhebungen. Nur Werkzeuge, welche die gewünschte Plattform und Programmiersprache unterstützen werden berücksichtigt. Tabelle 11.4 stellt ein Beispiel für solch eine Erhebung dar. Allerdings können solche Übersichten über verschiedene Werkzeuge sehr schnell ihre Aktualität verlieren. Daher sollte der Auswertende neue Erhebungen verwenden, z.B. die neueste Ausgabe von [Graham et al, 1995], oder Informationen aus dem World-Wide-Web, z.B. [APTEST].

[2] Andere Auswahl-Prozesse für Werkzeuge sind z.B. in [Dustin et al., 1999], oder [Kit, 1995] zu finden.

4. Als letztes vergleicht der Auswertende die vom Verkäufer gelieferten oder anderweitig ermittelten Informationen über das Werkzeug mit den gewichteten Auswahlkriterien. Er bewertet die Werkzeuge je nach Grad ihrer Erfüllung der Kriterien, und trägt die Daten in eine Datenbank ein. Er kann auch Daten aus früheren Bewertungen verwenden. Das Ergebnis der Auswertung ist eine Empfehlung von einem oder mehreren Werkzeugen für den gegebenen Test.

11.3 Arten von Testwerkzeugen

In jeder Phase des Software Entwicklungsprozesses werden unterschiedliche Testaktivitäten durchgeführt. Daher sind für jede Phase auch unterschiedliche Arten von Werkzeugen relevant.

Tabelle 11.2 gibt einen Überblick über Arten von Werkzeugen, zugeordnet nach Phasen im Software Entwicklungsprozess. Die Liste inkludiert alle Werkzeuge, die irgendwie Einfluss auf den Testprozess haben. Die folgenden Kapitel beschreiben dann im Detail die Arten von Testwerkzeugen für jede Phase im Testprozess.

Entwicklungsphase	**Werkzeugtyp**	**Beschreibung**
Alle Phasen	Werkzeuge zur Modellierung von Geschäftsprozessen	Erfassung von Benutzeranforderungen, Automatisierung der schnellen Konstruktion von flexiblen, graphischen Client-Server-Applikationen
	Configuration Management Tools	Unterstützen die Einrichtung von wichtigen Daten-Repositories
	Werkzeuge zur Fehlerverfolgung	Verwaltung von Fehlern aus allen Entwicklungsphasen
	Verwaltung von technischen Reviews	Erleichtern die Kommunikation und automatisieren den Prozess von technischen Reviews / Inspektionen
	Dokumentgeneratoren	Helfen bei der Erstellung von Dokumenten
	Werkzeuge zur Analyse der Testüberdeckung oder zur Codeinstrumentierung	Identifizieren noch ungetesteten Code und unterstützen dynamisches Testen
	Werkzeuge zur Messung der Usability	Erstellung von Anwenderprofilen, Taskanalysen, Prototyping und Walkthroughs
	Prototyping Tools	Prototyping der Applikation unter Verwendung von High-Level Programmiersprachen
	Stub-Generatoren	Erzeugung von Stub-Routinen
	Automatische Vergleicher	Aufzeigen von Unterschieden zwischen Dateien, Bildern, etc.

Entwicklungsphase	Werkzeugtyp	Beschreibung
	Simulatoren	Simulieren Applikationen u.a. zur Messung der Skalierbarkeit
Definition der Anforderungen	Werkzeuge zur Verwaltung von Anforderungen	Verwaltung und Organisierung der Anforderungen; unterstützen Entwurf von Testprozeduren; unterstützen Berichte über Testfortschritt; unterstützen Requirements-Tracking
	Anforderungsprüfer	Prüfung von Syntax, Semantik und Testbarkeit
	Use Case Generatoren	Erstellung von Use Cases
Analyse und Design	Werkzeuge für den Datenbankentwurf	Erstellung eines Modells der Datenbank, automatische Generierung der DB aus dem Modell und umgekehrt
	Werkzeuge für den Anwendungsentwurf	Definition der Softwarearchitektur; Objektorientierte Analyse, Modellierung, Design und Konstruktion
	Struktur-, Fluss- und Ablaufdiagramme	Unterstützen die Verwaltung von Prozessen
	Testskriptgeneratoren	Erzeugen Testskripts aus den Anforderungen oder dem Entwurf von Daten- und Objektmodellen, aus Ursache/Wirkungsgraphen oder dem Sourcecode
Implementierung	Syntaxprüfer / Debugger	Check der Syntax und Debugging; üblicherweise in Form einer IDE (Integrated Development Environment)
	Detektoren für Speicherlecks und Laufzeitfehler	Spüren Laufzeitfehler und Speicherlecks auf
	Werkzeuge für Test des Sourcecodes	Prüfung der Wartbarkeit, Portierbarkeit, Komplexität, zyklomatischen Komplexität, und der Einhaltung von Standards; erstellen Metrikdaten
	Statische und Dynamische Analysatoren	Bilden die Qualität und die Struktur des Codes ab
	Diverse Implementierungswerkzeuge	Abhängig von der Applikation, unterstützen u.a. Codegenerierung
	Unit Test Tool	Automatisierung von Modultests
Test	Werkzeuge zur Testverwaltung	Verwaltung des Tests

Entwicklungsphase	Werkzeugtyp	Beschreibung
	Testdatengeneratoren	Erzeugung von Testdaten
	Werkzeuge für Netzwerktests	Überwachung, Messung, Test und Diagnose der Performance über das gesamte Netzwerk
	GUI-Testwerkzeuge (Capture & Playback)	Automatisierte GUI-Tests; Capture / playback Werkzeuge nehmen Benutzereingaben auf, sodass sie automatisch abgespielt werden können
	Non-GUI Test-Treiber	Erlauben automatische Ausführung der Tests von Produkten ohne GUI
	Performance-, Last-, Stresstestwerkzeuge	Unterstützen Last-, Stress- und Performancetests
	Werkzeuge zum Test der Umgebung	Testwerkzeuge für diverse Testumgebungen, z.B. Windows, MVS, UNIX, X-Windows, und das World-Wide-Web

Tabelle 11.2: Überblick über Arten von Testwerkzeugen nach Entwicklungsphase [Dustin et al, 1992]

11.4 Werkzeuge für Planung und Verwaltung

Werkzeuge für die Phase Planung und Verwaltung sind hauptsächlich solche für die Verwaltung des strukturierten Testprozesses. Sie umfassen

- Fehlerverfolgung
- Testmanagement
- Planung
- Fortschrittsüberwachung
- Konfigurationsmanagement

Die Werkzeuge für das Konfigurationsmanagement, die Planung und die Fortschrittsüberwachung sind größtenteils die selben wie diejenigen für das Management des Softwareprojekts und werden deshalb hier nicht näher beschrieben.

11.4.1 Fehlerverfolgung

Werkzeuge zur Fehlerverfolgung (Bug Tracking Tools) unterstützen den Workflow für die Fehlerverwaltung, wie er in Kapitel 7 gezeigt ist. Im allgemeinen sind das typische Client/Server-

Applikationen, die aus einer zentralen Datenbank und Benutzerschnittstellen für Benutzer mit verschiedenen Rollen bestehen. Die Rollen umfassen meist Tester, Entwickler, Testmanager und Konfigurationsmanager. Oft sind die Clients entweder kleine Executables oder HTML-basierte Webclients.

Die Basisfunktionalität von Fehlerverfolgungswerkzeuge umfasst Erfassung, Modifizierung und Abfrage von Fehlermeldungen im Rahmen eines Arbeitsablaufs (*Workflow*). Zusätzliche Features beinhalten.

- Benachrichtigung der für den nächsten Schritt zuständigen Person vie E-Mail.
- Einbindung von Anlagen (*Attachments*) in Fehlermeldungen.
- Reports, Statistiken und Programme für Metriken.
- Vergleich von Projekten

11.4.2 Testmanagement

Diese Werkzeuge bieten ein integriertes Set an Funktionen im Bereich der Planung, Fortschrittsüberwachung, Verwaltung von Fehlermeldungen und Konfigurationsmanagment. Obgleich die Funktion für jeden Bereich meistens nicht so umfangreich ist wie bei einem spezifischen Werkzeug, besteht die Stärke eines Testmanagementwerkzeugs in der Integration zwischen den verschiedenen Bereichen [Pol et al, 2000].

Manche Testmanagementwerkzeuge bieten Funktionen für die Verwaltung von Anforderungen, Dokumenten, den daraus abgeleiteten Testfällen, und die Verwaltung der Ergebnisse jedes Testfalls.

Manche Anbieter vereinen verschiedene Werkzeuge in einer Suite; Testmanagementwerkzeuge helfen dann beim Umgang mit all den unterschiedlichen Werkzeugen. Dadurch unterstützen sie eine einfaches Zusammenspiel der Werkzeuge miteinander und fördern somit einen glatt verlaufenden Testprozess.

11.5 Werkzeuge für die Testfallspezifikation

Werkzeuge für die Phase der Testfallspezifikation können in zwei Kategorien eingeteilt werden: Testfallgeneratoren und Testdatengeneratoren. Testfallgeneratoren bilden logische Testfälle, während Testdatengeneratoren die Daten generieren, um aus logischen Testfällen konkrete zu machen.

11.5.1 Testfallgeneratoren

Aufbauend auf ein formales Modell der Applikation erzeugen Testfallgeneratoren Testfälle, die ein bestimmtes Überdeckungsmaß garantieren. Die meisten Testfallgeneratoren verwenden Black-Box Techniken, aber einige wenige Werkzeuge verwenden White-Box Techniken oder Hybridformen, die auf einer formalen Beschreibung der Anforderungen basieren.

Das benötigte Modell kann ein Graph (z.B. ein Datenflussdiagramm), eine formale Sprache (z.B. Pseudocode) oder der Sourcecode selbst sein. Die Definition eines solchen Modells ist meist keine einfache Aufgabe und erfordert erfahrene Tester.

Die Testfälle bestehen aus Eingabedaten (Eingabevektoren), aus Aktionsfolgen oder Kombinationen davon. Die Eingabevektoren können entweder aus Klassen von Eingabedaten bestehen oder aus konkreten Eingabedaten.

Im allgemeinen benötigen die erzeugten Testfälle eine manuelle Nachbearbeitung durch den Tester. Bei Eingabevektoren müssen die Methoden zur Eingabe in die Applikation durch den Tester festgelegt werden. Testfälle, die rein aus Aktionsfolgen (ohne konkrete Daten) oder auf Klassen basierenden Eingabevektoren bestehen, nennt man logische Testfälle. Diese erfordern, dass der Tester konkrete Eingabedaten definiert. Dieser Schritt kann durch Testdatengeneratoren unterstützt werden, wie sie im nächsten Kapitel beschrieben sind.

11.5.2 Testdatengeneratoren

Testdatengeneratoren unterstützen die Definition von konkreten Testfällen aus logischen Testfällen. Sie verwenden entweder Regeln für die Testdatenselektion, oder sie generieren die Daten zufällig. Letztere Methode ist nützlich für die Definition von Last- und Stresstests.

11.6 Werkzeuge für die Durchführung

Werkzeuge für die Phase Durchführung sind vielfältig und können in zwei Gruppen eingeteilt werden: solche für die Testausführung, und solche, die die Analyse von Testergebnissen unterstützen [Pol et al, 2000]. Tabelle 11.3 zeigt die Werkzeuge für beide Zwecke.

Werkzeuge für Testausführung	**Werkzeuge für die Ergebnisanalyse**
Funktionale Testwerkzeuge	Komparatoren
Performance Test Tools	Dynamische Analysewerkzeuge
Stubs und Driver	Werkzeuge zur Datenbankabfrage
Statische Analysewerkzeuge	Werkzeuge zur Systemüberwachung
Unterstützung der Fehlerbeschreibung	

Tabelle 11.3: Typen von Werkzeugen für die Testdurchführung

11.6.1 Funktionale Testwerkzeuge

Funktionale Testwerkzeuge werden auch oft Record / Playback Tools, GUI-Testtools oder Capture / Replaytools genannt. Die essentielle Funktion dieser Werkzeuge ist es, Aktionen an der Benutzerschnittstelle aufzuzeichnen, so dass sie später wiedergegeben werden können.

Während der Wiedergabe führen diese Werkzeuge eingebaute und/oder benutzerdefinierte Prüfungen durch, um das aufgezeichnete mit dem tatsächlichen Verhalten zu vergleichen. Zu diesem Zweck werden die Aktionen in Form eines Testskripts, das aus Aktionen und Ergebnisprüfungen besteht, aufgezeichnet. Die meisten funktionalen Testwerkzeuge bieten eine Skriptsprache, die eine vereinfachte Form einer Programmiersprache ist, z.B. von Basic oder C.

Die meisten modernen GUI-Testtools verwenden „widget" Technologie; das bedeutet, dass sie Elemente in Fenstern als Objekt mit Eigenschaften erkennen, unabhängig von der Position des Elements am Bildschirm. Die Aktionen in den Testskripts werden auf diesen Objekten durchgeführt. Die Methode der Objekterkennung bestimmt, welche Programmiersprachen unterstützt werden.

Die Basisfunktionalität der meisten Capture / Replaytools beinhaltet eine variable Geschwindigkeit der Wiedergabe, Pausen in der Testdurchführung, Logs der Testergebnisse und Komparatoren zur Erkennung und Darstellung von Abweichungen vom erwarteten Verhalten. Manche Capture / Replay Tools bieten eine Metasprache, z.B. um die Abarbeitungsreihenfolge der Testskripts zu bestimmen.

In den meisten Fällen sind die aufgezeichneten Skripts bloß lineare Sequenzen von Aktionen und Ergebnisprüfungen, die keinerlei Flusskontrolle beinhalten. Das macht sie sehr empfindlich gegenüber Änderungen im Programm. Um die Skripts flexibler zu machen, muss Flusskontrolle manuell eingebaut werden. Das erfordert Entwicklung, die einem konventionellen Softwareprojekt ähnlich ist. Je mehr die Applikation Änderungen unterworfen ist, desto höher ist der erforderliche Aufwand fürs Skripting.

Das Erstellen von Testskripts ist teurer als die manuelle Ausführung eines Tests. Daher ist automatische Testdurchführung nur profitabel, wenn der Test mehr als einmal durchgeführt wird, und wenn Programmänderungen nicht die Durchführung behindern. Die wenigsten Programmänderungen und die meisten Testiterationen treten bei Regressionstests auf. Daher sind Capture / Replaytools am wertvollsten für Regressionstests.

Funktionale Testwerkzeuge können auch folgendermaßen „missbraucht" werden: bei der Durchführung von explorativen Tests kann der Tester bei Vorhandensein eines solchen Werkzeugs alle Aktivitäten aufzeichnen. Wenn ein Fehler auftritt, kann der Tester nachsehen, was er genau gemacht hat, was es leichter macht, den Fehler zu reproduzieren. Der Einfachheit halber kann er dann das Skript als Anlage zur Fehlermeldung dazugeben.

11.6.2 Werkzeuge für Leistungstests

Werkzeuge für Leistungstests, auch Performance Test Tools genannt, werden zum Test des Zeitverhaltens der Applikation verwendet. Üblicherweise ist das Zeitverhalten gegeben durch die Antwortzeit von bestimmten zeitkritischen Funktionen. Es gibt zwei Hauptkategorien von Performance Test Tools:

- Werkzeuge zum Test der Einzelbenutzerperformance (oder einfach Performance Test Tools) sind hauptsächlich funktionale Testwerkzeuge, die zur Messung von Antwortzeiten verwendet werden.

- Werkzeuge für Last- und Stresstests (Load Test Tools) simulieren eine hohe Zahl von Anwendern, um die Applikation unter schwere Last zu setzen. In den meisten Fällen wird dabei das User Interface umgangen, d.h. die Daten werden direkt an die Schnittstellen geliefert.

Oft werden Performance Test Tools mit Werkzeugen zur Überwachung kombiniert, um Engpässe der Ressourcen zu finden.

11.6.3 Stubs und Driver

Stubs und Driver sind Programme, die anstelle von noch nicht vollendeten Systemkomponenten verwendet werden, wie Abbildung 11.2 zeigt. Sie werden hauptsächlich während Modul- und Integrationstests verwendet, solange das Gesamtsystem noch nicht verfügbar ist, aber auch nachher, um Tests z.B. durch Umgehung des GUI effizienter zu gestalten. Es gibt kaum standardisierte Testwerkzeuge von diesem Typ, da sie allzu abhängig von den Systemeigenschaften sind und in den meisten Fällen aus nur ein paar Zeilen Code bestehen. Dennoch sind Stubs und Driver ein wichtiges Hilfsmittel zur Erleichterung von Low-Level Tests, und werden daher in Softwareprojekten häufig verwendet.

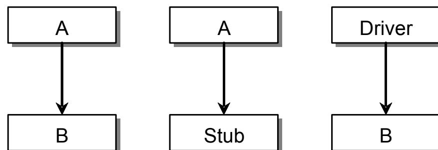

Abbildung 11.2: Stubs und Driver [Pol et al, 2000]

Stubs sind Programmmodule, die anstelle des echten Subsystems aufgerufen werden. Sie emulieren den Output des kompletten Subsystems. Meist liefern sie immer denselben Output. Sie werden so geschrieben, dass der gewünschte Output leicht geändert werden kann, z.B. durch Lesen des Outputs aus einem File. So können Testfälle für alle möglichen gültigen und ungültigen Outputdaten (die ja Input für das aufrufende Programm darstellen) rasch durchgeführt werden. Stubs sind bei der Top-Down Integration sehr häufig.

Driver sind Programmmodule, die anstelle des Komplettsystems ein Subsystem aufrufen. Sie emulieren den Input für das System in gleicher Weise wie Stubs den Output eines Subsystems. Driver finden während der Bottom-Up Integration Verwendung.

11.6.4 Statische Analysewerkzeuge

Statische Analysewerkzeuge führen Überprüfungen des Systems ohne Programmausführung durch. Dazu gehören z.B. Prüfungen der Datenbankkonsistenz, Prüfungen auf Einhaltung von Programmierrichtlinien und Style Guides, Analyse der Modulstruktur, etc. Statische Analysewerkzeuge sind ein wichtiges Hilfsmittel zum Test von Qualitätskriterien, die schwer oder unmöglich dynamisch zu testen sind, wie z.B. Wartbarkeit, Änderbarkeit, Testbarkeit etc.

Werkzeuge dieser Art sind oft in die Entwicklungsumgebung integriert. Sie können z.B. in ein Datenbankverwaltungsprogramm oder den Compiler integriert sein. Oft werden statische Analysewerkzeuge für im Projekt bekannte, häufig wiederkehrende Probleme auch selbst geschrieben. Auf dem Markt erhältliche stand-alone Werkzeuge leisten z.B. HTML-Validierung, Linktests oder API-Tests.

11.6.5 Unterstützung der Fehlerbeschreibung

Der Testprozess kann nur erfolgreich sein, wenn Fehler effizient gemeldet werden. Wenn wichtige Information fehlt, werden die Fehleranalyse und der Nachtest teurer. Daher werden Werkzeuge zur Unterstützung des Testers bei der Fehlerbeschreibung verwendet. Dies sind z.B. Werkzeuge für Screenshots, Analysewerkzeuge für die von der Applikation im Debug-Modus generierten Logs etc.

11.6.6 Komparatoren

Komparatoren helfen bei der Erkennung von Unterschieden zwischen zwei Objekten, z.B. Dateien, Screenshots, Speicherinhalte. Komparatoren werden sowohl manuell durch Tester verwendet, als auch automatisch von Testautomatisierungswerkzeugen, z.B. in der Anzeige des Fehlerprotokolls von funktionalen Testwerkzeugen.

11.6.7 Dynamische Analysewerkzeuge

Dynamische Analysewerkzeuge werden gemeinsam mit der zu testenden Applikation gestartet. Sie protokollieren gewisse Laufzeiteigenschaften der Applikation und bieten Mechanismen zur Analyse dieser Logs. Um möglichst genaue Informationen zu liefern, sind die meisten dynamischen Analysewerkzeuge auf eine Programmiersprache und –umgebung spezialisiert. Beispiele sind:

- Werkzeuge zur Ermittlung der Codeüberdeckung informieren über den Grad, zu dem die ausgeführten Testfälle die Softwarestruktur überdecken. Sie bieten nützliche Hilfe bei der Messung der Effizienz von Testfallspezifikationsmethoden.
- Detektoren für Speicherlecks versuchen, Speicherbereiche zu finden, die nach Gebrauch nicht freigegeben werden. Solche Fehler führen zu Instabilitäten in der Applikation und treten meist nur nach langem Gebrauch auf. Daher sind sie manuell sehr schwer zu finden.
- Engpassdetektoren geben eine Analyse der kumulierten Ausführungszeit aller in der Applikation aufgerufenen Funktionen. So können die zeitaufwendigsten Funktionen erkannt werden. Zusätzlich zeichnen sie einen Aufrufgraphen, der dabei hilft, unnötige Abhängigkeiten zwischen Funktionen zu finden.

11.6.8 Debugger

Wenn ein Fehler gefunden wurde, muss er auch lokalisiert werden. Werkzeuge, welche die mühsame Arbeit der Fehlerlokalisierung unterstützen, reichen vom gewöhnlichen Debugger mit Breakpoints und Überwachungsmechanismen bis hin zu automatisierten Debuggingtools, die Fehler finden und vielleicht sogar beheben können [Wotawa, 2001]. Diese Debugger basieren auf Prinzipien wie Program Slicing, Applikationsmodellen oder Mutationstesten.

11.6.9 Werkzeuge zur Datenbankabfrage

Oft führen Testfälle zu Änderungen in der Datenbank, die über die zu testende Applikation nicht direkt zugänglich sind. Um die Korrektheit dieser Änderungen zu prüfen, muss der Tester Zugriff auf die Datenbank haben. Viele Datenbanken bieten ihre eigenen Abfragewerkzeuge. Manche Abfragewerkzeuge verwenden standardisierte Schnittstellen wie ODBC.

Manche Abfragewerkzeuge bieten auch die Möglichkeit, Daten zu ändern, was sie nützlich für administrative Zwecke macht.

11.6.10 Werkzeuge zur Systemüberwachung

Werkzeuge zur Systemüberwachung, auch Monitoring Tools genannt, werden verwendet, um ein besseres Bild vom Verbrauchsverhalten des Systems zu erhalten. Daten über den Ressourcenverbrauch werden gesammelt und in einem Log abgelegt. Manche Werkzeuge erlauben die Überwachung des Ressourcenverbrauchs auf mehreren Rechnern gleichzeitig, z.B. durch Nennung ihrer IP-Adressen. Durch Analyse der Logs können Fehler oder Schwachpunkte wie Performanceengpässe, Speicherlecks oder Ressourcenverschwendung gefunden werden. Übliche Ressourcen für die Überwachung sind:

- CPU-Last
- Speichernutzung
- Netzwerklast
- Zugriff aufs Dateisystem

Marktüberblick über Testwerkzeuge Tabelle 11.4 basiert auf [Kit, 1995][3], mit Erweiterungen von [APTEST] und gibt einen Überblick über am Markt erhältliche Testwerkzeuge. In den Spalten der unterstützten Betriebssysteme steht H für Host, d.h. das Werkzeug wird auf diesem System installiert, und für unterstützt, d.h. das Werkzeug kann Aktivitäten auf diesem System automatisieren, läuft aber am Host. MF ist abgekürzt für Mainframe.

[3] Kit's Buch wurde zuerst in 1995 veröffentlicht, aber die Nachauflage von 1999 enthält eine neuere Liste von erhältlichen Tools aus 1998.

Typ	#	Werkzeug	Hersteller	Java	Win	Unix	MF
Test Planning, Management and Control							
Schedule Estimation	1	KnowledgePLAN	Software Productivity Research	✓	H	✓	✓
	2	SLIM	QSM	✓	H	✓	✓
Requirements Management	3	icCONCEPT RTM	Integrated Chipware, Inc.	✓	H	H	✓
	4	DOORS	QSS Inc.	✓	H	✓	✓
	5	Requisite Pro	Rational Software Corp.	✓	H	✓	✓
Test Management	6	QA Director	Compuware		H	H	✓
	7	SQA Manager	Rational Software Corp.		H		
	8	Test Director	Mercury Interactive Corp.		H		
Configuration Management	9	CA-Endevor	Computer Associates		✓	H	H
	10	Changeman	Serena International				H
	11	ClearCase	Rational Software Corp.		H	H	
	12	Continuus/CM	Continuus Software			H	
	13	PVCS Version Manager	Intersolv		H		✓
Problem Management	14	Continuus/PT	Continuus Software			H	
	15	Clear DDTS	Rational Software Corp.	✓	✓	H	✓
	16	PVCS Tracker	Intersolv		H		✓

Typ	#	Werkzeug	Hersteller	Java	Win	Unix	MF
Reviews & Inspections							
Technical Reviews Management	17	ReviewPro	Software Development Technologies	✓	H	H	✓
Complexity Analysis	18	Visual Quality	McCabe & Associates		H	H	✓
Code Comprehension	19	AcquaProva	CenterLine Development Systems			H	
	20	Visual Reengineering	McCabe & Associates		H	H	✓
Test Design & Development							
Database Generators	21	CA-Datamacs/II	Computer Associates				H
	22	TestBytes	Mercury Interactive Corp	✓	H	✓	✓
Test Execution & Evaluation							
Unit Testing	23	JavaSpec	Sun Microsystems, Inc.	H	✓	✓	✓
Java Capture/Playback	25	JavaStar	Sun Microsystems, Inc.	H	✓	✓	✓
Native Capture/Playback	25	QAHiperstation	Compuware		H		H
	26	QA Partner	Segue Software Inc.	✓	H	H	H
	27	QAPlayback	Compuware		H		H
	28	QC Replay	CenterLine Development Systems			H	
	29	SQA Robot	Rational Software Corp	✓	H		

Typ	#	Werkzeug	Hersteller	Java	Win	Unix	MF
	30	WinRunner	Mercury Interactive Corp	✓	H		✓
	31	XRunner	Mercury Interactive Corp			H	
Non-Intrusive Capture/Playback	32	Ferret	Azor, Inc.	✓	✓	✓	✓
	33	TestRunner/NI	Qronus Interactive	✓	✓	✓	✓
Web Capture/Playback	34	AutoTester Web	AutoTester Inc.		H		
	35	Silk	Segue Software Inc.		H		
	36	SQA Robot	Rational Software Corp		H		
	37	WebTest	Mercury Interactive Corp		H		
Coverage Analysis	38	JavaScope	Sun Microsystems, Inc.	H	✓	✓	✓
	39	Visual Testing	McCabe & Associates		H	H	✓
Memory Testing	40	BoundsChecker	Compuware		H		
	41	HeapAgent	MicroQuill Software Pub Inc.		H		
	42	Purify	Rational Software Corp.		H	H	
Client/Server	43	Automated Test Facility	Teradyne, Inc.		H		
	44	LoadRunner	Mercury Interactive Corp.		H	H	
	45	QualityWorks	Segue Software Inc.		H	H	✓
Performance/Simulation	46	LoadRunner	Mercury Interactive Corp.		H	H	
	47	PerformanceStudio	Rational Software Corp.			H	

Typ	#	Werkzeug	Hersteller	Java	Win	Unix	MF
	48	TPNS	IBM				H

Tabelle 11.4: Überblick über am Markt erhältliche Werkzeuge für den Softwaretest

Legende: H = Host, ✓ = unterstützt

11.7 Kriterien für funktionale Testwerkzeuge

Dieses Kapitel gibt einen Überblick über Werkzeugkriterien, die bei der Auswahl von funktionalen Testwerkzeugen berücksichtigt werden müssen. Wie in Kapitel 11.3 beschrieben, ist die Selektion von Testwerkzeugen anhand von Kriterien sehr wichtig, um Testautomatisierung erfolgreich einzuführen. Dies gilt besonders für funktionale Testwerkzeuge. Eine weite Bandbreite von funktionalen Testwerkzeugen mit verschiedenen Eigenschaften ist am Markt erhältlich (siehe Kapitel 0)

Die in diesem Kapitel beschriebenen Kriterien können die Grundlage für einen Auswahlprozess wie in 11.3 sein. Der Auswahlprozess selbst ist jedoch sehr aufwendig, sowohl was Zeit als auch Geld betrifft. Die Ergebnisliste hängt stark von der zu testenden Applikation ab, und entwickelt sich mit der Zeit.

11.7.1 Entwicklungsumgebungen und Programmiersprachen

Die meisten funktionalen Testwerkzeuge unterstützen ein Betriebssystem und ein schmales Band an Programmiersprachen auf dieser Plattform. Die Entwicklungsplattform der zu testenden Applikation bestimmt, welche Testwerkzeuge überhaupt in Frage kommen. Testwerkzeuge, die eine große Anzahl Programmiersprachen unterstützen, unterstützen oft keine davon gut.

Objekterkennung

Methoden. Funktionale Testwerkzeuge müssen Objekte erkennen, um Testskripts generieren zu können. Unter den Methoden für die Objekterkennung versteht man, wie ein Objekt eindeutig identifiziert wird, und dass es als Objekt des richtigen Typs erkannt wird. Sie bestimmen die Flexibilität und Erweiterbarkeit des funktionalen Testwerkzeugs. Erkennungsmethoden sind beispielsweise eine interne Windows-ID oder der Objektname.

Dieses Kriterium ist eng verbunden mit den unterstützten Entwicklungsumgebungen, da die Effektivität einer gegebenen Erkennungsmethode mit der Programmiersprache variiert. Benutzerdefinierte Objekt sind die größte Herausforderung ans Werkzeug. Oft ist die einzige Möglichkeit, dieses Kriterium zu testen, das funktionale Testwerkzeug auszuprobieren.

Erweiterbarkeit. Entwicklungsumgebungen entwickeln sich weiter, und neue Typen von Objekten können auftauchen. Ein funktionales Testwerkzeug muss entweder eine manuelle Erweiterung der Objekttypenbibliothek unterstützen, oder der Anbieter muss regelmäßig neue Versionen davon zur Verfügung stellen.

Skriptsprache

Um funktionale Testwerkzeuge effizient einsetzen zu können, muss die Skriptsprache Kommandos zur Flusskontrolle, z.B. „if" und „for" unterstützen. Im allgemeinen sind Skriptsprachen, die mehr Möglichkeiten bieten (deren Verwendung aber nicht fordern!) besser als Sprachen mit eingeschränkten Möglichkeiten.

Datengetriebene Tests

Im Skript hart codierte Daten für Eingaben oder Ergebnisprüfungen machen es sehr unflexibel und empfindlich gegenüber Änderungen im getesteten System. Daher müssen Daten extern gespeichert werden können, z.B. in einer Datenbank. Funktionale Testwerkzeuge sollten also Wege bieten, um Daten für Eingaben und Ergebnisprüfungen von einer Datenbank zu lesen. Manche Testwerkzeuge unterstützen solche datengetriebenen Tests direkt. Wenn nicht, müssen sie zumindest Kommandos für den Datenbankzugriff in der Skriptsprache zur Verfügung stellen.

Ergebnisprüfungen

Um Vergleiche zwischen dem beabsichtigten und dem tatsächlichen Verhalten des Systems anzustellen, muss das Werkzeug Ergebnisprüfungen in den Skripts aufnehmen. Solche sind z.B. Prüfungen von Objekteigenschaften, der Existenz eines Objekts oder von Speicher- und Dateiinhalten. Die Vielseitigkeit der zur Verfügung gestellten Ergebnisprüfungen bestimmt wiederum die Flexibilität des Testwerkzeugs.

Wie bei der Objekterkennung können mit der Zeit auch neue Typen von Ergebnisprüfungen notwendig werden. Dafür muss das Testwerkzeug zumindest die Möglichkeit bieten, externe, benutzerdefinierte Ergebnisprüfungen einzubinden, z.B. aus einer DLL oder einem Shared Object.

Fehlerprotokoll

Funktionale Testwerkzeuge müssen bei der Testausführung ein Fehlerprotokoll erzeugen. Nur dann kann eine große Anzahl an Tests ohne Überwachung durchgeführt werden. Die meisten Werkzeuge bieten auch einen Log-Viewer, der Testfälle mit Fehlern hervorhebt, und in denen Komparatoren eingebaut sind, um Fehlerursachen schnell finden zu können.

Verhalten bei Fehlern

Das Verhalten im Falle von unerwarteten Fehlern im zu testenden System – z.B. unerwarteten offenen Fenstern, Systemabstürze oder Timingprobleme – variiert von Werkzeug zu Werkzeug. Manche Werkzeuge bieten die Möglichkeit, das Verhalten in den Optionen anzupassen; manche reagieren auf Fehler immer gleich. Das richtige Verhalten hängt vom System und vom Fehler ab; daher sind diejenigen Werkzeuge besser, die mehr – und intelligentere – Optionen bieten. Wenn die eingebauten Optionen nicht ausreichen, muss ein Error-Handling manuell eingebaut werden, was den Skriptingaufwand erhöht.

Wartbarkeit

Dieses Kriterium beschreibt, wie einfach Testskripts an Änderungen in der zu testenden Applikation angepasst werden können. Die Wartbarkeit hängt von der Menge der hart codierten Eingabe- und Ergebnisdaten ab, bzw. von der Struktur dieser Daten im Fall von datengesteuerten Testwerkzeuge. Die kritischsten Änderungen an der Applikation sind solche am Workflow oder

der Datenstruktur. Die meisten funktionalen Testwerkzeuge kommen mit kleineren Änderungen an der Benutzeroberfläche selbst zurecht.

Automatische Generierung von Testskripts

Manche funktionalen Testwerkzeuge bieten die Möglichkeit, Skripts aus einer formalen Darstellung der Testfälle zu generieren. Meist sind das diejenigen Werkzeuge, die auch die Testfälle aus einem formalen Applikationsmodell erzeugen (siehe Kapitel 11.5.1). Solche Werkzeuge benötigen keine Datensteuerung, weil Änderungen an den Testfallbeschreibungen durchgeführt, und daraus die Skripts neu generiert werden. Im allgemeinen leiden die erzeugten Skripts unter mangelnder Flexibilität.

Integration von Monitoring Tools

Bei der Protokollierung von Testergebnissen ist es sinnvoll, Ressourceninformationen gleich mitzuprotokollieren. Manche funktionalen Testwerkzeuge unterstützen die Integration von Monitoring Tools, die das Verbrauchsverhalten während der automatischen Durchführung von Tests protokollieren.

Vervielfältigung von Testskripts

Manche Testtypen, z.B. Last- und Stresstests erfordern eine große Menge von Testfällen, die gleichzeitig oder hintereinander ausgeführt werden. Einige funktionale Testwerkzeuge ermöglichen die Vervielfältigung von Testskripts mit nach zufälligen oder regelbasierten Methoden diversifizierten Eingabedaten.

Meta-Sprache

Eine Meta-Sprache in einem funktionalen Testwerkzeug erlaubt es dem Tester, die Abfolge der durchzuführenden Testfälle zu bestimmen. In manchen Fällen können sogar Abhängigkeiten zwischen Testfällen modelliert werden, sodass die Ausführung eines Testfalles nur stattfindet, wenn gewisse andere ohne Fehler durchgeführt wurden.

Timing-Funktionen

Die automatische Durchführung von Testfällen ist besonders nützlich wenn sie zu einer gewissen Zeit automatisch gestartet werden kann. So kann ein Test beispielsweise über Nacht laufen gelassen werden, um die Last am Server oder Netzwerk untertags zu verringern.

Non-intrusives Testen

Wenn das funktionale Testwerkzeug auf derselben Hardware wie die getestete Applikation läuft, beeinflusst das zu einem gewissen Grad die Performance der Applikation. Non-intrusives Testen bedeutet, dass das Testwerkzeug mit keinem oder vernachlässigbarem Einfluss auf die Applikation durchführt. Dies kann z.B. durch Abzweigen des Outputs zum Vergleich auf einem anderen Rechner geschehen, und durch Wege, die Applikation von außen zu steuern, ohne die für die Applikation wichtigen Ressourcen anzugreifen.

11.8 Testentwicklung für die Automatisierung funktionaler Tests

Dieses Kapitel behandelt das Basisparadigma für die Automatisierung von funktionalen Tests unter Verwendung von Capture / Replay Tools. Es geht dann auf die Probleme mit diesem Prozess und viel versprechende Lösungsstrategien ein. Das Kapitel basiert hauptsächlich auf [Dustin et al, 1992] und Kaner, 1997.

11.8.1 Basisparadigma

Bei der Verwendung von funktionalen Testwerkzeugen wird üblicherweise ein Testskript pro Testfall aufgezeichnet, und nach Änderungen in der Applikation eine Untermenge aller Testfälle wiedergegeben. Das Werkzeug vergleicht einige Daten in der Applikation mit aufgezeichneten Daten, und produziert ein Fehlerprotokoll (*Fehler-Log*), das durch den Tester ausgewertet wird.

Allerdings bietet dieser Ansatz nicht genug Flexibilität und so gut wie keinen Vorteil gegenüber manuellen Tests – man denke nur an gewünschte Änderungen an der Applikation oder der Datenbank. Dustin et al. [Dustin et al, 1992] schlagen eine modulare Testskriptarchitektur vor, sowie eine Reuse-Datenbank, um bessere Ergebnisse zu erzielen.

Der Testdesigner entwickelt eine modulare Testskriptarchitektur, die den Eigenschaften der zu testenden Applikation entspricht. Testskripts werden nach der Aufzeichnung dieser Architektur angepasst, d.h. ein Entwicklungsschritt wird eingeführt. Das Ergebnis ist eine Menge von modularen Skripts, die in anderen Tests wieder verwendet werden können. Die Verwendung einer Reuse-Datenbank erleichtert die Wiederverwendung.

Der Tester gibt sodann die modifizierten Testskripts wieder und wertet die Ergebnisse aus, unter Umständen öfters für Regressions- oder andere Tests. Die Entwicklungsaktivitäten werden, mit Ausnahme des Entwurfs der Testskriptarchitektur, für jeden Testfall durchgeführt. Alle Aktivitäten werden ihrerseits für jede zu testende Applikation wiederholt, und für jedes verwendete Testwerkzeug. Abbildung 11.3 zeigt den kompletten Prozess.

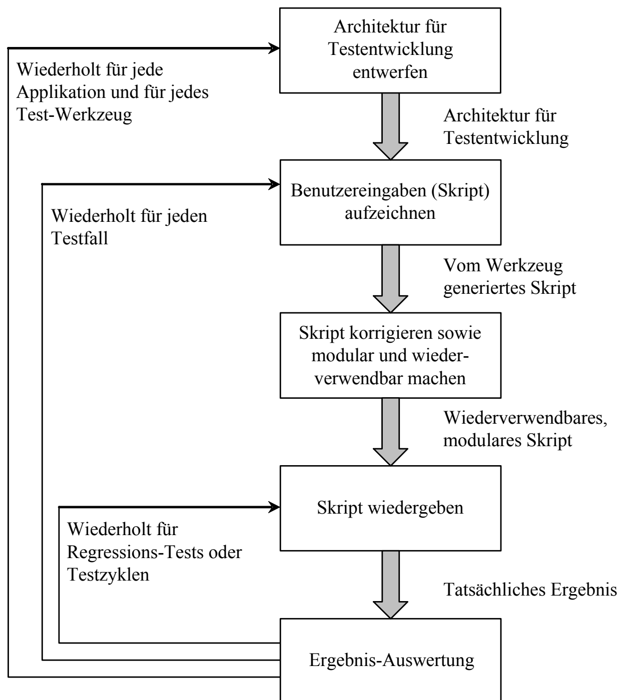

Abbildung 11.3: Entwicklungsaktivitäten mit Capture / Replay Tools, basierend auf [Dustin et al, 1992]

Test Skripts sollten so früh wie möglich entwickelt werden, d.h. parallel mit der Applikation. Wenn das richtig gemacht wird, können sie Testaktivitäten während des gesamten Entwicklungsprozess unterstützen. Tabelle 11.5 zeigt den Zusammenhang zwischen der Anwendungsentwicklung und der Testskriptentwicklung [Dustin et al, 1992].

Phase	Entwicklungsprozess	Testprozess
Modulentwicklung	Modul aus den Anforderungen entwerfen	Test planen und Testumgebung einrichten
	Modul implementieren	Testfälle und Testdaten spezifizieren
	Modul debuggen	Testskripts schreiben oder anhand des Moduls aufzeichnen
	Modultest durchführen	Testskript durch Laufen am Modul debuggen. Verwendung von Werkzeugen für Modultests.
	Fehler korrigieren	Testskripts als Regressionstest nach Korrekturen laufen lassen

Phase	Entwicklungsprozess	Testprozess
Integration	Module zum System zusammenbauen. Integrationstest mit verbundenen Modulen ausführen. Fehlermeldungen reviewen	Modultest Skripts kombinieren und neue Skripts, die Modulverbindungen zeigen, hinzufügen. Verwendung von Werkzeugen für Integrationstests.
	Fehler korrigieren	Testskripts als Regressionstest nach Korrekturen laufen lassen
Systemtest	Fehlermeldungen reviewen	Testskripts wo möglich zu Prozeduren auf Systemebene integrieren und zusätzliche Prozeduren entwickeln. Systemtest durchführen und Ergebnisse aufzeichnen.
	Fehler korrigieren	Testscripts als Regressionstest nach Korrekturen laufen lassen
Abnahmetest	Fehlermeldungen reviewen	Untermenge der Systemtestprozeduren als Teil des Abnahmetests ausführen.
	Fehler korrigieren	Testskripts als Regressionstest nach Korrekturen laufen lassen

Tabelle 11.5: Die Verbindung zwischen Entwicklung und automatisiertem Testprozess [Dustin et al, 1992]

11.8.2 Probleme mit dem Basisparadigma

Trotz der Verwendung einer modularen Testskriptarchitektur und einer Reuse-Bibliothek verbleiben einige Probleme im Basisansatz, die in diesem Kapitel beschrieben werden. Mögliche Lösungsansätze werden im nächsten Kapitel behandelt.

Hart codierte Testdaten

Wenn Testprozeduren über ein Testwerkzeug aufgezeichnet werden, bindet das Werkzeug automatisch hart codierte Werte ein. Aufzeichnen von Testprozeduren, die auf hart codierte Werte bauen, begrenzt die Wiederverwendbarkeit und die Wartbarkeit der Testprozedur:

- Ein heute als Konstante aufgezeichnetes Datumsfeld kann morgen das Skript in seiner korrekten Funktion behindern.
- Zwei Testfälle mit verschiedenen Werten in einem Feld, z.B. das Geburtsdatum eines Kunden, würden zwei Testprozeduren implizieren, selbst wenn die verbleibenden Teile des Testfalles gleich sind.

- Wenn Änderungen an der Applikation durchgeführt werden, z.B. ein anderer Wertebereich für ein Feld, müssen alle betroffenen Testdaten und daher alle betroffenen Testskripts angepasst werden.
- Wenn ein Fenster irgendwelche variablen Daten in seinem Namen anzeigt, z.B. das Systemdatum, dann ist das Testwerkzeug unter Umständen nicht in der Lage, den korrekten Wiedergabekontext zu finden.

Ansätze für dieses Problem sind die Verwendung von Wildcards, von globalen Konstanten, und einer datengesteuerten Architektur.

Änderungen am Workflow der Applikation

Selbst kleine Änderungen am Workflow der zu testenden Applikation, z.B. Aufruf einer Funktion über einen Button anstelle eines Menüeintrags, können eine große Menge an Testfällen unbrauchbar machen. Dieses Problem resultiert daraus, dass die Abläufe genauso wie die Testdaten in der Skriptprozedur hart codiert sind.

Mögliche Lösungen zu diesem Problem sind die Entwicklung von kontextunabhängigen Skripts oder die Verwendung einer frameworkbasierten Architektur.

Änderungen an oder Inkompatibilitäten mit Funktionen des Testwerkzeugs

Probleme mit Funktionen des Testwerkzeugs können entstehen, wenn eine neue Version des Testwerkzeugs eingesetzt wird, oder wenn Inkompatibilitäten mit der zu testenden Applikation existieren.

- Eine Änderung der Syntax einer Funktion kann eine Änderung aller Skriptprozeduren, die diese Funktion verwenden, nötig machen. Eine frameworkbasierte Architektur mit Kapselung jeder einzelnen Funktion des Testwerkzeugs in eine Skriptfunktion kann dieses Problem lösen.
- Inkompatibilitäten mit der zu testenden Applikation, z.B. fehlerhafte Erkennung eines Objekts, können zu Fehlern in Funktionen des Testwerkzeugs führen. Dieses Problem kann gelöst werden, indem man eine frameworkbasierte Architektur verwendet und jede Funktion der Applikation in einer Skriptfunktion kapselt.

Wiederverwendbarkeit

Wenn jeder Testfall als eigene Skriptprozedur implementiert wird, so ist dieses Skript mit hoher Wahrscheinlichkeit nicht wieder verwendbar, weil es zu spezifisch ist. Je allgemeiner und kürzer eine Skriptprozedur ist, desto höher sind ihre Chancen auf Wiederverwendung. Allerdings müssen kurze und allgemeine Skripts irgendwie kontrolliert und mit spezifischen Werten versorgt werden, z.B. über Shell Skripts oder Wrapper-Funktionen.

Dieses Problem ist mit der Verwendung einer modularen Architektur bereits teilweise gelöst, und es kann durch Verwendung einer Framework basierten Architektur weiter verringert werden.

Start des Tests

Wenn keine Shell Skripts oder Wrapper-Funktionen eingesetzt werden, muss das Testteam manuell sicherstellen, dass die zu testende Applikation sich im korrekten Anfangszustand befindet

und dass die Testprozeduren in korrekter Reihenfolge abgearbeitet werden. Die nötige Reihung der Testprozeduren könnte in jeder Testprozedur als Vorbedingung für die Ausführung dokumentiert werden. Dieser manuelle Ansatz wirkt jedoch den Zielen der Automatisierung entgegen.

Eine mögliche Lösung für dieses Problem ist, kontextunabhängige Testprozeduren zu schreiben.

Fehlerbehandlung

Viele Testprozeduren werden ohne Berücksichtigung der Fehler, die während des Tests auftreten können, geschrieben. Das kann zu Problemen führen, z.B. wenn die Tests ohne Beaufsichtigung laufen und ein Test zu einem Absturz der Applikation führt; ein folgender Test kann dann nicht durchgeführt werden, wenn keine Mechanismen zur Wiederherstellung vorhanden sind. Die gesamte Test-Suite wird in diesem Fall versagen.

Um eine Test-Suite wirklich automatisiert zu machen, muss eine Fehlerbehandlung eingebaut werden, welche die verschiedenen Fehlertypen, denen das Skript begegnen kann, berücksichtigt.

11.8.3 Strategien zur Problemlösung

Die folgenden Strategien sind verbreitete Praktiken, um die Effizienz von funktionaler Testautomatisierung zu erhöhen. Der Testentwickler kann diese je nach Anforderungen der Applikation kombinieren.

Modulare Testskripts

Ein modulares Skript erhöht die Wartbarkeit. Ein kleineres Testskript ist einfacher zu verstehen und zu debuggen. Das Skript in logische Module zu zerlegen ist daher ein Weg, komplexe Skripts zu bewältigen. Wenn das Testteam den logischen Fluss der Skripts in der Planungsphase korrekt bestimmt, wird aus jedem dort identifizierten logischen Block ein Modul. Zusätzlich kann durch die Modularisierung die Erstellung der Skripts auf mehrere Testentwickler aufgeteilt werden.

Wenn eine Testprozedur modular entworfen wurde und die zu testende Applikation sich ändert, dann muss das Testteam nur die betroffenen modularen Komponenten anpassen. Daher müssen Änderungen für gewöhnlich an nur einer Stelle durchgeführt werden.

Jedes Modul kann aus mehreren kleinen Funktionen bestehen. Eine Funktion umfasst mehrere Zeilen Skript, die eine Aufgabe ausführen. Beispielsweise führt eine Funktion `Login()` folgende Aktionen aus:

1. Start der Applikation
2. Eingabe der der User-ID
3. Prüfung der User-ID (Fehlerprüfung)
4. Eingabe des Passworts
5. Prüfung des Passworts (Fehlerprüfung)
6. Drücke OK.

Anstatt lange Abfolgen von Aktionen in der selben Testprozedur zu inkludieren, sollten Testprozeduren eher kurz und modular sein. Sie sollten sich auf ein bestimmtes Testgebiet konzentrieren, wie z.B. eine einzige Eingabemaske, oder auf eine verwandte Menge wiederkehrender Aktionen, wie z.B. Navigation oder Fehlerprüfungen. Für umfangreichere Tests können modulare Testprozeduren einfach von anderen Testprozeduren aufgerufen werden. Diese wiederum zu geordneten Top-Level Prozeduren (Wrappers oder Shell-Prozeduren) zusammengefasst werden.

Noch einmal die Vorteile von modularen Testprozeduren auf einen Blick:

- Modulare Testprozeduren können von überdachenden Shell-Prozeduren aufgerufen, in diese hineinkopiert oder zu ihnen zusammengefügt werden.
- Sie können einfach angepasst werden, wenn Entwickler Änderungen an der zu testenden Funktion durchführen.
- Sie sind einfacher zu debuggen.
- Änderungen müssen nur an einer Stelle durchgeführt werden, was „kaskadierende“ Effekte umgeht.
- Die Wartung von modularen Tests ist einfacher.

Testprozeduren sollten so entworfen werden, dass sie als allein stehende Tests fungieren können, d.h. dass der Output eines Tests nicht als Input für den nächsten dient. Mit einer solchen Strategie können modulare Skripts in jeder Testprozedur in beliebiger Reihenfolge eingesetzt werden.

Datengesteuerte Architektur

Ein Weg, wieder verwendbare und wartbare Testprozeduren zu entwickeln ist, einen datengesteuerten Ansatz zu verwenden. Dabei werden Daten entweder von einer Datei oder Datenbank gelesen, anstatt sie hart codiert im Testskript zu haben, oder dorthin geschrieben.

Im allgemeinen ist es zu empfehlen, dass der Testentwickler Datenwerte in den Testprozeduren nicht hart codiert. Um Fehler bei der Ausführung zu verhindern, muss er solche hart codierten Werte durch Variablen ersetzen und diese, wann immer möglich und machbar, mit Daten aus einem Flat File oder einer Datenbank befüllen.

Dadurch wird auch die Wartung der Daten erleichtert, und Skripts gewinnen an Wiederverwendbarkeit. Jedes mal, wenn die Testprozedur mit einem anderen Set an Daten wiedergegeben werden soll, muss nur das Datenfile geändert werden.

Ein weiterer Schritt ist die Auslagerung von Informationen über den Workflow. Nicht nur Eingabedaten werden von einem File oder einer Datenbank gelesen, sondern auch Objekte, Kommandos und erwartete Ergebnisse. Dadurch wird der Skriptcode in höchstem Maße von den Daten getrennt, denn Informationen über Objekte, Kommandos und Ergebnisse stellen auch Daten dar. Dies minimiert den Aufwand für Skriptmodifikationen und Wartung. Die Funktionalität der Applikation ist in einer Tabelle dokumentiert, wie auch schrittweise Anweisungen für jeden Test. Tabelle 11.6 zeigt ein Beispiel mit Kommandos des SQA Robot von Rational.

Window (VB Name)	Window (Visual Text)	Control	Action	Arguments
StartScreen	XYZ Savings Bank	-	SetContext	-
PrequalifyButton	Prequalifying	PushButton	Click	-
frmMain	Mortgage Prequalifier	-	SetContext	-
frmMain	File	-	MenuSelect	New Customer

Tabelle 11.6: Beispiel-Tabelle zur Generierung automatischer Testskripts [Dustin et al, 1992]

Ein Testskript-Framework oder ein einfaches Parsing-Programm kann die Schritte aus der Tabelle lesen, bestimmen, wie jeder Schritt auszuführen ist, und ihn mitsamt Fehlerprüfungen auf Basis der gelieferten Fehlercodes ausführen. Ein geeignetes Skript-Framework führt den Test direkt aus; ein Parser extrahiert Daten aus der Tabelle und generiert daraus eine oder mehrere Testprozeduren, die als gesonderter Schritt durchgeführt werden.

Das Testteam kann eine GUI-Map anlegen, die Einträge für jeden Typ von GUI-Objekten, die im Test vorkommen, enthält. Jeder Eintrag in der GUI-Map enthält Information über den Typ des Objekts, das Fenster, in dem das Objekt vorkommt, sowie Größe und Position im Fenster. Jeder Eintrag enthält einen eindeutigen Identifier, der im Skript ähnlich wie die Strings zur Objekterkennung verwendet wird.

Frameworkbasierte Architektur

Ein Framework trennt die zu testende Applikation von den Testskripts durch eine Menge an Funktionen, die Aktionen in der Applikation kapseln, in einer Bibliothek. Ersteller von Testskripts behandeln diese Funktionen wie Befehle der Programmiersprache des Testwerkzeugs.

Ein Framework kann beispielsweise eine Funktion `openfile(p)` beinhalten, die eine Datei p öffnet. Sie kann so vorgehen, dass sie das Menü „Datei“ öffnet, das Kommando „Datei öffnen“ auswählt, den Dateinamen in das entsprechende Feld einträgt und den OK-Button drückt. Die Funktion kann auch mehr beinhalten, kann z.B. prüfen, ob die Datei p auch wirklich geöffnet wurde, oder den Versuch, die Datei zu öffnen, mitprotokollieren. Oder die Funktion reicht den Dateinamen und den Pfad einfach an das API des Programms. Die Funktion kann sich wöchentlich ändern: dem Ersteller von Skripts ist das egal, so lange `openfile(x)` die Datei x öffnet.

Frameworks beinhalten mehrere Typen von Funktionen. Man muss beachten, dass nicht alle Kommandos zur selben Zeit hinzugefügt werden können. Man muss Prioritäten setzen und die Bibliothek mit der Zeit aufbauen. Hier sind ein paar Basistypen bzw. Schritte:
Definiere jedes Feature der Applikation. Man kann Funktionen schreiben, die eine Menüauswahl durchführen, einen Dialog öffnen, den Wert einer Variablen setzen oder ein Kommando absetzen. Wenn die Benutzerschnittstelle eine dieser Aktionen ändert, ändert man die Funktion. Jedes Skript, das diese Funktion verwendet, passt sich automatisch an die Änderungen an.

Definiere Kommandos oder Features der Werkzeug-Sprache. Man kann eine zusätzliche Schicht einführen, indem man eine Funktion (Wrapper genannt) um jedes Kommando der Sprache des Testwerkzeugs legt. Ein Wrapper ist eine Routine, die um eine andere Funktion herum gebaut wird

und unter Umständen nichts anderes tut als die umwickelte Funktion aufzurufen. Man kann den Wrapper modifizieren um Funktionalität hinzuzufügen oder wegzunehmen, um einen Bug im Testwerkzeug zu umgehen, oder um einen Vorteil aus einem Update der Skriptsprache zu ziehen.

Definiere kleine, konzeptuell zusammengehörende, häufig ausgeführte Tasks. Das ist geradlinige Wiederverwendung von Code, die in der Testautomatisierung genauso wünschenswert ist wie in jedem anderen Softwareprojekt.

Definiere größere, komplexere, in mehreren Tests verwendete Testfälle. Es kann hilfreich erscheinen, größere Kommandosequenzen zu kapseln. Damit sind allerdings Risiken verbunden, vor allem, wenn man übertreibt. Eine sehr komplexe Sequenz wird wahrscheinlich nicht in vielen Skripts wieder verwendbar sein, so dass es den Aufwand nicht Wert ist, sie zu verallgemeinern, zu dokumentieren und Fehlerbehandlungen einzubauen, wie man es von einer komplett implementierten Funktion der Bibliothek erwarten würde. Außerdem steigt mit größerer Komplexität der Sequenz die Wahrscheinlichkeit, dass sie angepasst werden muss, wenn sich das UI ändert. Selten benutzte komplexe Kommandos können die Wartungskosten der Bibliothek dominieren.

Definiere Utility-Funktionen. Man kann beispielsweise eine Funktion zum standardisierten Logging von Testergebnissen schreiben. Jedes Werkzeug liefert ein Set an vorgegebenen Utilityfunktionen. Die Menge der zusätzlichen Funktionen muss also nicht sehr groß sein.

Fehlerbehandlung

Die Standards für die Entwicklung von Testprozeduren müssen berücksichtigen, dass Fehlerprüfungen dort im Skript eingebaut werden müssen, wo Fehler am häufigsten auftreten. Fehler müssen von derjenigen Testprozedur geliefert werden, die den Fehler entdeckt und weiss, was es für ein Fehler is. Der Einbau von Fehlerbehandlungsmechanismen, welche die wahrscheinlichsten Fehler abfangen, erhöht die Wartbarkeit und die Stabilität der Testprozeduren.

Testskripts können um Logik erweitert werden, die im Fehlerfall zu einem anderen Skript verzweigt, oder ein Skript aufruft, das einen Fehlerzustand bereinigt. Die erzeugte Fehlermeldung sollte für den Testentwickler selbsterklärend sein. Dieser Ansatz erleichtert das Debuggen einer Testprozedur, weil die Fehlermeldung das Problem direkt festlegt. Die Wege, wie Testprozeduren mit Fehlern umzugehen haben, zusammen mit Beispielen, sollten als Richtlinien (*Guidelines*) festgelegt werden.

Das folgende Testskript, das in SQA Basic entwickelt wurde, der Skriptsprache von SQA Robot von Rational, prüft die Existenz einer Datei. Wenn das Skript die Datei nicht findet, liefert es eine Fehlermeldung, dass die Datei nicht gefunden werden konnte, und beendet die Prozedur. Wenn das Skript die Datei findet, wird eine dementsprechende Meldung ins Log geschrieben (Assertions sind genauso wichtig für die Wartbarkeit wie Fehlermeldungen) und das Skript setzt fort.

```
' Existenz einer Datei prüfen
DataSource = SQAGetDir(SQA_DIR_PROCEDURES) & "CUSTPOOL.CSV"
If Dir(DataSource) = "" Then
    MsgBox "Cannot run this test. File not found: " & DataSource
    Result = 0
    Exit Sub
Else
    WriteLogMessage "Found " & DataSource
End-If
```

Kontextunabhängigkeit

Um Testprozeduren kontextunabhängig zu machen, müssen sie im selben initialen Zustand (z.B. Einstiegsfenster) beginnen und auch dort wieder enden. Dieser Ansatz repräsentiert den idealen Weg, modular zu programmieren, denn solche Skripts sind unabhängig vom Ergebnis, was bedeutet, dass das Ergebnis eines Tests nicht das Ergebnis eines anderen beeinflusst, und dass Daten und Kontext nicht beeinflusst werden. Das erhöht die Auswechselbarkeit der Testprozeduren, was den Einsatz von Shell-Prozeduren ermöglicht. Kontextunabhängigkeit wird durch die Einhaltung von Modularitätsprinzipien erleichtert.

Wildcards

Manche Testwerkzeuge erlauben die Verwendung von „wildcards". Wildcards wie z.B. „*" ermöglichen es dem Testskript, anstatt exakten Vergleichen solche mit gewissen Freiheitsgraden durchzuführen.

Wenn Testwerkzeuge verwendet werden, um Testprozeduren aufzuzeichnen, kann beispielsweise eine Wildcard eingesetzt werden um die Caption eines Fensters zur Identifikation des Fensters zu ermitteln. Bei Einsatz von z.B. SQA Robot, identifiziert das Kommando `Window SetContext`, in welchem Fenster darauf folgende Aktionen durchgeführt werden sollen. Dieses Kommando kann mit Wildcards durchgeführt werden, da viele Fenster im Titel einen variablen Anteil wie z.B. einen Dateinamen haben, der in vielen Fällen für die Wahl des Fensters nicht relevant ist.

Testentwicklung ist Software-Entwicklung!

Bei der Entwicklung von Testprozeduren muss der Testentwickler berücksichtigen, dass Testentwicklung dasselbe ist wie „normale" Softwareentwicklung. D.h. dass, zusätzlich zu den obigen Strategien, auch Basisprinzipien der Softwareentwicklung gelten, wie z.B.:

- Verwende Schleifen und Bedingungen
- Verwende globale Konstanten
- Definiere globale Funktionen

- Definiere Standards und Richtlinien und halte dich daran (Fehlerbehandlung, Kommentare, Dokumentation, kosmetische Standards, etc.)
- Folge dem Phasenmodell

11.9 Zusammenfassung

Testautomatisierung ist die Verwendung eines Testwerkzeugs, um Tests zu erleichtern. Ein Testwerkzeug ist ein Hilfsmittel, das eine oder mehrere Testaktivitäten unterstützt. Auf der einen Seite verlangt Testautomatisierung einen strukturierten Testprozess. Auf der anderen Seite hilft Testautomatisierung bei der Einrichtung eines strukturierten Testprozesses.

Testwerkzeuge haben sowohl Vor- als auch Nachteile. Die Auswahl des richtigen Werkzeuge ist die Grundlage für den Erfolg der Testautomatisierung. Der Prozess der Werkzeug-Auswahl muss mit großer Umsicht erfolgen und besteht aus vier Schritten:

- Analyse der Benutzeranforderungen
- Ermittlung der Werkzeugauswahlkriterien
- Suche nach Werkzeugen
- Bewertung der Werkzeuge

Testwerkzeuge können in jeder Phase des Testprozesses zum Einsatz kommen. Kommerzielle Testwerkzeuge für beinahe jede Testaktivität sind am Markt erhältlich.

Testentwicklung für funktionale Testwerkzeuge ist eine genauso komplexe Aufgabe wie jede andere Softwareentwicklung. Es existieren viele verschiedene Strategien zum erfolgreichen Einsatz, die ein Testentwickler den Anforderungen entsprechend kombinieren kann. Die ideale Strategie ist, modulare Skripts mit einer datengesteuerten, frameworkbasierenden Architektur und Fehlerbehandlungsmechanismen zu schreiben.

11.10 Literaturreferenzen

[Asböck, 2001] Asböck, Stefan: "Load testing for eConfidence", Segue Software Inc, 2001, ISBN 3-000-07359-0.

[ATR, 2003] Automated Testing Resources: http://www.sqa-test.com/ATS_Tips.html.

[ATT, 2003] Applied Testing and Technology, Inc.: "Automated and Manual Software Testing Tools and Services", http://www.aptest.com/.

[Chang et al, 1999] Chang, J.; Richardson D. J.: "ADLscope: an Automated Specification-based Unit Testing Tool", Software-Engineering-Notes, vol. 24, no.6; Nov. 1999, Pages 285-302.

[Chang, 2000] Chang Liu: "Platform-independent and tool-neutral test descriptions for automated software testing", Proceedings of the 2000 International Conference on Software Engineering, Pages: 713 -715.

[Changet al, 2000] Chang Liu; Richardson, D.J.: "Using application states in software testing", Proceedings of the 2000 International Conference on Software Engineering, Pages: 776.

[CQAC, 2003] Compuware QACenter: http://www.compuware.com/.

[Dalal et al, 1999] Dalal, S.R.; Jain, A.; Karunanithi, N.; Leaton, J.M.; Lott, C.M.; Patton, G.C.; Horowitz, B.M.: "Model-based testing in practice", Proceedings of the 1999 International Conference on Software Engineering, 1999, Pages: 285 -294.

[Dustin et al, 1992] Elfriede Dustin, Jeff Rashka, John Paul: "Automated Software Testing", Addison-Wesley, 1999, ISBN 0-201-43287-0.

[Ferguson et al, 1995] Ferguson, R.; Korel, B.: "Software test data generation using the chaining approach", Proceedings of the International Test Conference, 1995, Pages: 703 -709.

[Graham et al, 1995] Dorothy Graham et al.: "Computer-Aided Software Testing: The CAST Report", 1995.

[JUNIT, 2003] JUnit Testing Resources For Extreme Programming: http://www.junit.org/.

[Kaner, 1997] Kaner, Cem: "Pitfalls and Strategies in Automated Testing", Computer, vol.30, no.4, April 1997, Pages: 114 - 116.

[Kaner et al., 1999] Kaner, Cem; Falk, Jack; Nguyen, Hung Quoc: "Testing Computer Software", Wiley, 1999, ISBN 0-471-35846-0.

[Kasik et al, 1996] Kasik, David J.; George, Harry G.: "Toward automatic generation of novice user test scripts", Conference proceedings on Human factors in computing systems, April 1996.

[Kolish et al, 1999] Kolish, Tony; Doyle, Tom: "Gain eConfidence: The e-Business Reliability Survival Guide", Segue Software Inc., 1999.

[Kit, 1995] Edward Kit: "Software Testing in the Real World", Addison-Wesley, 1995, ISBN 0-201-87756-2.

[Lutsky, 1995] Lutsky, P.: "Automating testing by reverse engineering of software documentation", Proceedings of the 2nd Working Conference on Reverse Engineering, 1995, Pages: 8 -12.

[Mayrhauser et al, 1994] von Mayrhauser, A.; Mraz, R.; Walls, J.; Ocken, P.: "Domain based testing: increasing test case reuse", Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1994, Pages: 484 -491.

[Mayrhauser et al., 1996] von Mayrhauser, A.; Shumway, M.; Ocken, P.; Mraz, R.: "On domain models for system testing", Proceedings of the Fourth International Conference on Software Reuse, 1996, Pages: 114 -123.

[Mayrhauser et al, 1999] von Mayhauser, A.; Scheetz, M.; Dahlman, E.; Howe, A.E.: "Trade-offs in planner representation for automated software testing", Proceedings of the 1999 IEEE Aerospace Conference, Volume: 5, Pages: 83 -92.

[Memon et al, 2000] Memon, Atif M.; Pollack, Martha E.; Soffa, Mary Lou: "Automated test oracles for GUIs", ACM SIGSOFT Software Engineering Notes, Proceedings of the 8th

international symposium on Foundations of software engineering for twenty-first century applications, November 2000, Volume 25 Issue 6, Pages 30 - 39.

[Memon et al, 2001]Memon, A.M.; Pollack, M.E.; Soffa, M.L.: "Hierarchical GUI test case generation using automated planning", IEEE Transactions on Software Engineering, Volume: 27 Issue: 2 , Feb. 2001, Pages 144 -155.

[Michael et al, 1998] Michael, C.; McGraw, G.: "Automated software test data generation for complex programs", Proceedings of the 13th IEEE International Conference on Automated Software Engineering, 1998, Pages: 136 -146.

[MIET, 2003] Mercury Interactive Enterprise Testing: http://www.mercuryinteractive.com/.

[NPI, 2003] Newport Group, Inc.: "Justifying the Purchase of an Automated Testing Tool": http://www.newport-group-inc.com/PDF/!mercury.pdf.

[Pol et al, 2000] Martin Pol, Tim Koomen, Andreas Spillner: "Management und Optimierung des Testprozesses: ein praktischer Leitfaden für Testen von Software, mit TPI und TMap", dpunkt.verlag, 2000, ISBN 3-932-58865-7.

[Poston, 1992] Poston, R.M.; Sexton, M.P.: "Evaluating and selecting testing tools", IEEE Software, Volume: 9 Issue: 3 , May 1992, Pages: 33 -42.

[Poston, 1995] Poston, R.M.: "Testing tools combine best of new and old", IEEE Software, Volume: 12 Issue: 2, March 1995, Pages: 122 -126.

[RSQA, 2003] Rational SQA Suite Test Studio: http://www.rational.com/products/tstudio/index.jsp.

[Toeppe et al, 1999] Toeppe, S.; Ranville, S.: "Model driven automatic unit testing technology tool architecture", Proceedings of the 18th Digital Avionics Systems Conference, 1999, Volume: B.6-6 vol.2, Pages: 10.A.4-1 -10.A.4-11.

[SSF, 2003] Segue Silk family: http://www.segue.com/.

11.11 Übungen und Fragen

1. Was ist Testautomatisierung?
2. Nennen Sie Vor- und Nachteile von Testautomatisierung.
3. Beschreiben Sie, wie Sie ein für Ihr Projekt geeignetes Test-Werkzeug in effizienter Weise finden.
4. Welche Arten von Testwerkzeugen gibt es und was tun sie?
5. Welches sind die relevanten Kriterien zur Auswahl von funktionalen Testwerkzeugen?
6. Beschreiben Sie, wie funktionale Testwerkzeuge praktischerweise verwendet werden.
7. Welche Probleme können beim Einsatz von funktionalen Testwerkzeugen auftreten? Welche Lösungsansätze gibt es?

8. 1. Erstellen einen Katalog von Kriterien für die Auswahl von
   a. Performance Testwerkzeugen
   b. Testfallgeneratoren
   c. Testdatengeneratoren
   d. statischen Analyse-Werkzeugen
   e. dynamischen Analyse-Werkzeugen

9. Nehmen Sie an, Sie entwickeln eine Internet-Applikation zum Verkauf von Büchern. Als eines der relevantesten Qualitätskriterien wurde die Leistung des Web-Servers (Anforderungen/sec, Verfügbarkeit) und des Datenbankservers (Transaktionen/sec, Datenmengen im Terabyte-Bereich) identifiziert. Erstellen Sie eine Liste derjenigen Werkzeuge, die zur Unterstützung des Tests dieses Kriteriums in Frage kommen.

10. Erstellen Sie mit Java (http://java.sun.com/) ein Programm, das feststellt, ob ein eingegebenes Dreieck (Eingabe: drei Seitenlängen) rechtwinklig, gleichschenklig, gleichseitig, ein allgemeines oder gar kein Dreieck ist. Alternativ dazu können Sie auch ein anderes (vergleichbar einfaches) Java-Programm verwenden. Definieren Sie dann einen Modultest für das Programm und automatisieren Sie diesen mit JUnit (http://www.junit.org/).

11. Erstellen Sie eine Architektur (Datenbankmodell reicht) für die Entwicklung funktionaler Tests für eine Applikation und ein funktionales Test-Werkzeug Ihrer Wahl.