TOOLS FOR AUTOMATIC FUNCTIONAL TESTING

Seminararbeit aus
Testen von Software Systemen

Thomas Schöller 0157163 / 880
Markus Roth 9455467 / 881

WS 2004

Inhaltsverzeichnis

1 Einleitung

Um Applikationen zu testen, müssen alle möglichen Benutzereingaben und deren Kombinationen in Betracht gezogen und durchgeführt werden. Da die manuelle Realisierung dieser Tests einen zu großen Aufwand für eine oder mehrere Personen darstellen würde, werden Tests benötigt, die einen solchen Benutzer der Applikation simuliert (d.h. Maus Klicks und Eingabe).
Ein weiterer Aspekt ist, dass bei neuen Versionen der zu testenden Applikationen sichergestellt werden muss, dass die alte Funktionalität nicht verloren geht. Da aufgrund der enormen Anzahl an möglichen Kombinationen nicht alle Test Cases durchführbar sind, müssen einige repräsentative Fälle ausgewählt werden.
Beim manuellen Testen von Applikationen – auch mithilfe einer Liste von Test Cases – ist es nicht so leicht, alle Aktionen derart auszuführen, wie es beim letzten Test geschehen ist. Deshalb kann auch nicht gewährleistet werden, dass man das gleiche Ergebnis bzw. die gleiche Reaktion der Applikation wie beim letzten Test vorliegen hat.
Grund hierfür ist, dass bei Aktionen wie beispielsweise Maus Klicks durch eine reale Person, die exakte Durchführung nicht immer gegeben ist (z.B.: „Klicke auf Pos 340,240“ oder “Klicke 100mal im Abstand von je 1 Sekunde.“).
D.h. Menschen können bei weitem nicht - sowohl in der Eingabepräzision als auch in ihrem zeitlichen Verhalten - die Testbedingungen so präzise verifizieren wie es eine Simulation zustande bringt.
Eine andere Möglichkeit besteht darin, dass die Test Cases keine direkten Werteingaben definiert haben, sondern nur die Funktion oder den Use Case, der ausgeführt werden soll. Beispielsweise die Eingabe einer negativen Zahl in ein Eingabefeld: Welche Zahl eingegeben werden soll, ist nicht notwendigerweise definiert. Es besteht zwar die Option derartige Eingabewerte in den Test Cases zu definieren, bei komplizierteren Fällen jedoch ist dies oft nicht mehr möglich.

Fazit: Es ist notwendig, Tools für das Testen von Applikationen zu verwenden.

Weitere Vorteile von Tools sind:

- Dauertests sind einfach möglich
- Wiederverwendung von gleichen Testteilen
- Testlauf auch nachts und an Wochenenden möglich

Kriterien von Test Tools:

- Durchführung aller möglichen Benutzereingaben
- Loggen aller durchgeführten Aktionen
- Wiederverwendung von Test Code ermöglichen (Funktionen, Libraries, … definieren können)
- Spezielle Funktionen zum Zugriff auf typische Applikationselemente (Menüs, Pop Ups, Checkboxen, Comboboxen)

2 WIN BATCH

Das Tool stammt von der Firma Wilson WindowWare, erreichbar unter http:\\www.winbatch.com. WinBatch ist ein scriptbasierendes Programm, das eine Scriptsprache besitzt, die wie ein Batch-Programm – also vom ersten Befehl bis zum letzten – abgearbeitet wird. Das Starten von Threads, die Aktionen ausführen, ist allerdings nicht möglich.
Im Gegensatz zu normalen Batch-Programmen sind in der Scriptsprache Funktionen integriert, mit denen Mausbewegungen und –klicks, Tastatureingaben und Menübefehle simuliert werden können. Zusätzlich stehen noch eine Menge an Standardbefehlen, wie z.B.: File- und Stringfunktionen, Prozessmanagement, Benutzerinteraktion, ... zur Verfügung.

2.1 Funktionsweise von WinBatch:

WinBatch interpretiert ScriptDateien mit der Fileextension „*.wbt“. Dafür wird der Datei WinBatch.exe das wbt-File übergeben. Fehler im Script werden erst beim Ausführen der Zeile erkannt und am Ende zeigt WinBatch einen Fehlerdialog an.
Dies ist wichtig zu wissen, da beim Auftreten eines Fehlers der Fehlerdialog bestätigt werden muss und der Testablauf in der Zwischenzeit steht. Wenn nun der Test in der Nacht durchgeführt wird, braucht man immer jemanden, der die etwaigen Fehler bestätigt; bis dahin kommt es natürlich zu langen Wartezeiten.
WinBatch besitzt keinen automatischen Log der Aktionen, die ausgeführt wurden. Es muss alles selbst über Fileaktionen oder ähnlichem im Script gemacht werden.
z.B. (Pseudocode):

```
file_open(filename)
file_write(filename, “testcase=testergebnis”)
file_close(filename)
```

2.2 Fehlermanagement:

WinBatch verfügt über drei Error Modes zwischen denen via Scriptbefehl hin- und hergeschaltet werden kann.
Bei echten Scriptfehlern, bei denen ein Befehl nicht verstanden wird, bricht WinBatch ab - egal in welchem Errormode man sich befindet.
Bei allen anderen Fehlern, die WinBatch wirft, wie z.B.: fehlendes File beim Fileopen, wird der Fehlerdialog angezeigt.
Damit nun, trotz dem aufgetretenen Fehler, der Test weiterläuft, indem beispielsweise der nächste Testcase anspringt oder irgendwelche Codes aufgrund des Fehlers aufruft, kann der Error Mode so eingestellt werden, dass keine Fehlerdialoge mehr erscheinen.

2.3 Variablenmanagement

Alle Identifier, denen im Script ein Wert zugewiesen wird, gelten als Variablen. Es gibt keine Datentypen. Wird ein Variablenwert mit Hochkomma zugewiesen, wird er als String interpretiert. Außer bei selbstdefinierten Funktionen sind alle Variablen global bekannt (global scope).

2.4 Prozessmanagement:

Es können vom Script andere Programme, also auch WinBatch Scripts gestartet werden. WinBatchScripts können auch andere Scripts aufrufen, die im gleichen Interpreter laufen [Call(…)]. Dadurch wird zuerst das gerufene Script ausgeführt, und danach bei der Abarbeitung des ersten weitergemacht.
Die Programme können auf verschiedenste Weise gestartet werden:

a) Script wartet auf Beendigung des gestarteten Programmes [z.B.: RunWait(….)] oder es wartet eben nicht [z.B. Run(…)]
b) das gestartete Programmfenster wird nicht angezeigt (RunHide)
c) das gestartete Programmfenster wird minimiert (RunIcon)

Dadurch ist es möglich, einen Hauptablauftest zu schreiben, der alle anderen Tests initialisiert. Außerdem können auch DOS-System-Befehle abgesetzt werden.

2.4.1 Script Control Statements:

Natürlich gibt es alle Standard-Statements zum Definieren von Schleifen wie For, While, Break, If, Else, Switch.
Zusätzlich gibt es noch Labels im WinBatch. Diese können entweder mit Goto erreicht werden oder mit GoSub, wobei hier die Rücksprungsadresse gespeichert wird und mittels Return zurückgesprungen werden kann.

2.4.2 Benutzerdefinierte Funktionen

Es gibt zwei Arten von selbstdefinierten Funktionen - mit und ohne Zugriff auf die Variablen des aufrufenden Scripts.
#DefineFunction hat keinen externen Variablenzugriff und alle definierten Variablen haben eine Lebensdauer bis zum Ende der Funktion. Eine Funktion kann maximal 16 Parameter übergeben bekommen.
Um die Funktion benutzen zu können, muss der Interpreter mindestens einmal über die Definition gelaufen sein.
Die Funktion kann auch in einem eigenen Script File liegen, das mittels „Call“ geladen wird. So kann eine Art Library Script erstellt werden.
#DefineSubRoutine funktioniert genauso, nur dass es einen globalen Variablenscope und deshalb Zugriff auf alle Variablen hat.

2.4.3 Parameterübergabe an das Script:

Es besteht die Option, dem Script Commandline-Parameter zu übergeben.
Diese müssen getrennt durch Leerzeichen dem Scriptfile angehängt werden.
Der Aufruf des Programmes sieht dann folgendermaßen aus:
winbatch.exe script.wbt param1 param2
Im Script kann mittels der vordefinierten Konstante “param0”, die Anzahl der übergebenen Commandline-Parameter abgefragt werden. Die Parameter sind dann als Konstanten „param1“ bis „paramN“ abrufbar.

2.4.4 Programmabbruch

Mittels den Befehlen Exit und Terminate kann das Script beendet werden. Bei Terminate kann noch eine Message angegeben werden, die beim Beenden angezeigt wird.
Von außen kann das Script nur durch Abschießen der winbatch.exe beendet werden, was sich unter Umständen als schwierig erweisen kann.

2.4.5 Dynamische Befehle

Mittels Execute kann ein übergebener String als WinBatchbefehl abgesetzt werden. Dadurch ist ein dynamisches Erstellen eines Befehls während der Scriptabarbeitung möglich.

2.5 Debug

WinBatch verfügt über die wichtigsten Debugmöglichkeiten, wie Setzen von Breakpoints, Beobachten von Variablenwerten, Steppen durch das Script.
Der Debug Mode kann während der Scriptabarbeitung verändert werden.

2.6 Maus Handling:

Maus-Aktionen wie move, click etc. werden mit absoluten Pixel-Positionen gemacht.

2.7 Windowmanagement:

… ist wohl die wichtigste Funktionalität für das Testen von Applikationen.
Fenster werden bei WinBatch über deren "Title" erkannt.Es funktioniert auch, wenn man nur einen Teil des Titles angibt; das Fenster wird dennoch erkannt.
Dies hat insofern eine große Bedeutung, da viele Applikationen – hauptsächlich Editoren – die geladenen Files oder Projekte an den Titel anhängen.
Wäre das Aufrufen mittels partiellen Titles nicht möglich, könnten keine dynamischen Testfunktionen geschrieben werden.
Mit Fenstern können alle Aktionen duchgeführt werden, die das Operationssystem (z.B.: Windows) zulässt:

- Move
- Resize
- Minimize/ Maximize

2.8 Zeitmanagement:

Normalerweise versucht WinBatch immer, die Aktionen so schnell wie möglich auszuführen. Da jedoch manche Aktionen der Applikation längere Zeit benötigen als beispielsweise der in folge durchzuführende Mausklick, muss eine Wartezeit eingeführt werden. Dies ist nötig, damit die Reihenfolge der Aktionen bzw. die Ausführung der Aktionen selbst sichergestellt ist.
Hierfür gibt es in WinBatch eine TimeDelay-Funktion. Die Länge der Wartezeit muss allerdings vom Ersteller des Scripts selbst herausgefunden werden.
Neben der TimeDelay-Funktion gibt es auch noch die Yield( )-Funktion, die die CPU für andere Applikationen als WinBatch freigibt.

2.9 Benutzereingaben:

WinBatch kann nicht nur zum Testen von Applikationen genutzt werden, sondern auch um generelle Aufgaben zu automatisieren.
Bespiel: bestimmte Files mit einer Auswahl an Fileextensions auf ein Directory kopieren.
Hier sollte das Quellverzeichnis bei jedem Start des Batches angegeben werden, ohne immer das Script umschreiben zu müssen. Dafür gibt es Message-Funktionen, die während dem Scriptablauf angezeigt werden, um Inputs vom User zu fordern; im angegebenen Beispiel wäre dies ein Filechoose-Dialog.

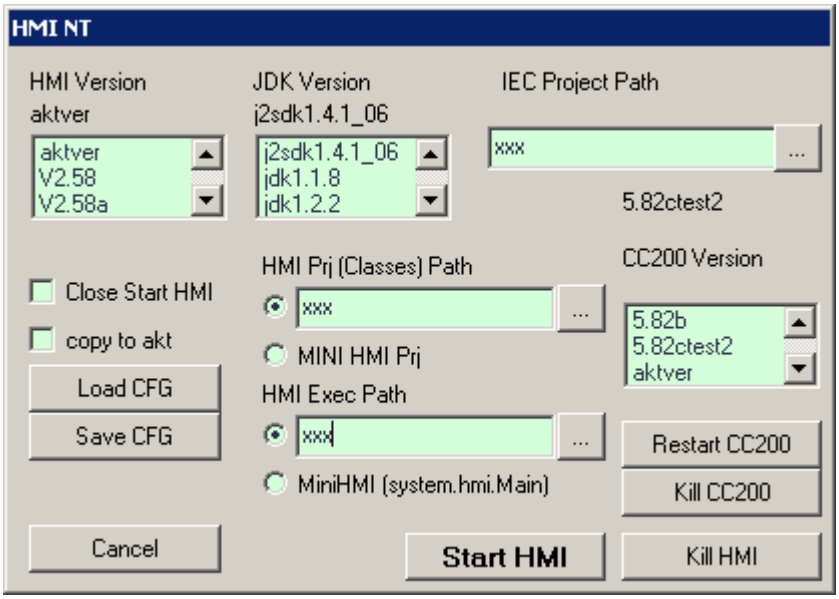
2.10 WinBatch Input:

- Message: MessageBox mit Ok-Button; nur zur Information für den User
- YesNoMessage: MessageDialog mit Yes- und No-Buttons
  Gewählter Button kann durch Returnwert der Message abgefragt werden
- AskLine: Dialog mit einem abfragbaren Edit-Feld zur Eingabe von Zeichen
- Standarddialog: wie AskDirectory, AskFileName, AskFont, AskColor, …
- AskTextBox: Dialog mit Eingabefeld mit mehreren Linien
- ….

2.11 Benutzerdefinierter Dialog:

Wenn die bisher genannten Dialoge nicht ausreichend oder für spezielle Aufgaben andere Eingabefelder von Nöten sind, kann man auch selbst Dialoge definieren.
Dazu bietet WinBatch einen eigenen Dialog-Editor an, in dem leicht Dialoge gezeichnet werden können.
Als Dialog-Elemente stehen die folgenden zur Auswahl:

- Button auch mit Bild statt Button-Text
- Radio-Button, auch Gruppenbildungen möglich
- CheckBox
- Bilder
- Eingabezeilen und –felder
- Static Textlabel
- Variable Textlabel
- Spinner
- ComboBox
- ListBox
- FileListBox
- Calendar
- GroupBox



Auf allen Control-Elementen können die typischen visuellen Attribute wie

- Invisible
- Disable
- ReadOnly
- Password
- Font
- Background
- Color

verändert werden.
Die so erstellten Dialoge können entweder als wdl-File gespeichert und im Script-File geladen werden oder der den Dialog beschreibende Code kann ins Clipboard kopiert und direkt ins Script-File eingefügt werden.
Wird vom Script ein Dialog aufgerufen, so wird im Script angehalten, bis dass eine Taste gedrückt worden ist. Diese Taste kann - wie bei den Standarddialogen – durch den Rückgabewert des Dialogabrufs abgefragt werden, wobei nun die Zahl des Rückgabewerts einer selbst definierten Zahl des Control-Buttons entspricht.
Nun kann eine Logik aufgebaut werden, die entsprechend des gedrückten Buttons, die jeweils zugeteilte Aktion ausführt.
Auf diese Weise können leicht komfortable Suiten von Testautomatisierung und halbautomatische Testabläufe aufgebaut werden.
Es gibt aber wesentliche Nachteile bei den WinBatch-Dialogen:
Man kann sie nicht minimieren und im WinBatch-Window sind keine Ausgaben möglich. Deshalb muss der Dialog immer im sichtbaren Bereich bleiben.

2.12 Zeitfunktionen:

Neben der schon besprochenen TimeDelay-Funktion gibt es noch eine Menge von Funktionen zum Formatieren, Vergleichen, Setzen und Holen von Zeiten, die für's Testen wichtig sein können.
Da es sehr oft relevant ist, *wann* ein Fehler aufgetreten ist oder ein Testergebnis eingetragen wird, muss es eine Funktion geben, die die aktuelle Systemzeit ausgibt. In WinBatch gibt es hierfür die

Funktion GetExactTime( ), welche die aktuelle Zeit formatiert ausgibt, und GetTickCount, die die Anzahl der Ticks seit dem Start von Windows ausgibt.
TimeDate gibt das aktuelle Datum aus.
Es gibt außerdem noch eine Reihe von Funktionen, die einen Zeitunterschied in verschiedener Formatierung (Unterschied generell, in Sekunden, in Tagen) berechnen.
Weiters können alle Filezeiten ausgelesen und verändert werden.

2.13 Clipboard-Funktionen:

Da viele Applikationen Clipboard-Funktionen wie

- Copy
- Cut
- Paste
- …

haben und diese für die Automatisierung oft nicht gut zu gebrauchen sind, gibt es im WinBatch standardmäßig folgende wichtige Funktionen:

- ClipGet: liest den Clipboardinhalt in einen String aus
- ClipGetEx (format): gibt den Inhalt des Clipboards in dem angegebenen Format zurück. Als Formate kommen in Frage:
  CF_text: normaler ANSI
  CF_OEMTEXT: Text mit Zeichen aus dem OEM-Zeichensatz
  CF_UNICODETEXT: Unicode-Text

Damit können fast alle Cut- und Copy-Funktionen getestet werden. Um für den Test von Paste völlig testapplikationsunabhängig zu sein, ist die Funktion ClipPut( ) von Vorteil, die einen String in das Clipboard kopiert.
Für komplexere Clipboardformate gibt es noch die Funktionen BinaryClipGet und BinaryClipPut.
Sollte dies auch nicht genügen, müssen die Clipboard-Funktionen der Testapplikation indirekt getestet werden. Beispiel:
Text in der Applikation mit Cut oder Copy ins Clipboard holen, den Text in der Applikation umschreiben und nach einem Paste entweder den Text direkt vergleichen (wenn es einen gespeicherten File gibt, in dem man suchen kann) oder einen weiteren Umweg machen, indem man den Clipboardinhalt in ein anderes Tool (z.B.: Notepad) pastet, wo man mittels Filevergleich die Verifizierung vornehmen kann.

2.14 Disk-Funktionen:

DiskExist testet, ob ein Drive funktioniert. Dies ist bei Applikationen wichtig, die auf gemountete Laufwerke zugreifen können sollen. DiskFree (freier Speicher), DiskInfo, DiskScan (Liste von Disks), DiskSize (totaler Speicher), … sind Funktionen, die nötig sind, wenn man den genauen Systemzustand und damit die Testbedingungen auslesen oder man Speicherprobleme und Veränderungen im Disk-Management der Testapplikation herausfinden will.

2.15 File-Funktionen:

WinBatch besitzt sämtliche Funktionen, die man zum Schreiben, Lesen, Kopieren, Verschieben, Vergleichen und Lesen von Informationen über das File wie Attribute, braucht.
Schreibe- und Lesebefehle gibt es auch für einen binären Buffer, von dem die Daten gelesen bzw. in den die Daten geschrieben werden können.

FileItemize gibt eine Liste von Files zurück, die mit TABs getrennt sind. Als Eingabe wird ein String mit Files, die entweder mit Wildcard (z.B.: *.exe) versehen oder mit „|“ voneinander getrennt sind (z.B.: notepad.exe|explorer.exe) oder eine Kombination von beiden darstellen (z.B.: *.exe|*.bat).

Damit können Kopierfunktionen auf einzelne Extensions leicht durchgeführt werden.
Nachteil dabei ist, dass nicht in Subdirectories gesucht wird. Dies muss mittels DirItemize (siehe Dir-Funktionen) selbst implementiert werden.

2.16 Dir-Funktionen

Zusätzlich zu den Standard-Funktionen zum Wechseln, Erstellen, Löschen, Umbenennen, Abfragen des aktuellen Directory, Ändern der Directory Attribute (read only, archive, system, hidden) oder Abfragen der Directorygröße, gibt es noch ein paar spezielle Funktionen, die sich gut fürs Testen verwenden lassen.
Das bereits angesprochene DirItemize funktioniert so wie das FileItemize, nur dass es keine Liste von Files, sondern ein Liste von Directories unter dem aktuellen Directory zurückgibt.
DirHome liefert den Pfad des angeführten Script Files.

2.17 Funktionen aus DLLs

Durch DLL-Funktionen können Funktionen aus DLLs benutzt werden. Damit kann man direkt auf die Windows-API zugreifen oder einen Extender (siehe später) schreiben.

2.18 Extender

Zusätzlich gibt es noch offizielle Extender, um die Funktionalität für eine bestimmte Funktionsgruppe zu erweitern.
Damit ist es z.B. möglich, Buttons direkt anzusprechen, serielle Schnittstellen zu benutzen, Netzwerksockets sehr einfach zu benutzen, ….
Aktuelle Liste verfügbarer Extender:

- Light Ascii Flat File DataBase Extender
- File Search Extender
- CpuInfo Extender
- Control Manager Extender
- EHLLAPI Terminal Emulator Extender
- Huge Math Extender
- Pixie Image Extender
- WinInet Extender
- IP Address Grabber Extender
- MAPI Extender
- Networking Win32 Extenders
- ODBC Extender
- Process Information Extender
- Printer Control Extender
- Postie Extender
- RAS Connectoid Extender
- Registry Search Extender
- Serial Port Extender
- Shell Operations Extender
- WinSock Extender
- Zipper Extender

3 WinRunner

3.1 Einleitung

WinRunner von der Firma "Mercury" ist im Gegensatz zu WinBatch als volles Applikationstesttool anzusehen und auch als solches entwickelt worden. Es hat einige Features, die WinBatch nicht hat und die für das Testen - speziell von komplizierteren Applikationen - benötigt werden.
Wie WinBatch ist WinRunner ein Script-basierendes Tool, das das Script zur Laufzeit interpretiert. Die Scriptsprache nennt sich TSL (Test Script Language) und ist im wesentlichen wie jene von WinBatch aufgebaut; allerdings mit einem geringeren Umfang.
Zusätzlich zum Interpreter, der auch als Editor fungiert, läuft noch ein Server, der sich zwischen Rechner und Benutzereingaben hängt und alle Eingaben des Benutzers mitprotokolliert. Dadurch wird es ermöglicht, dass Eingaben aufgenommen und wieder abgespielt werden können.

3.2 WinRunner Testing Modes

Es gibt bei WinRunner zwei Arten wie ein Test aufgenommen bzw. abgespielt werden kann.

1) Context Sensitive Mode

Dabei werden die aufgenommenen Aktionen mit Windows und Elementen in Windows (z.B.: Buttons, CheckBoxen, ...) in eine GUI-Map eingetragen.
Es handelt sich hierbei um Files, die eine Liste von Windows und Windowselementen beinhalten. Jeder Eintrag enthält eine eindeutige Beschreibung des Elements. Diese Beschreibung besteht aus einer minimalen Liste von Eigenschaften des Elements, sodass es von den anderen Elementen im gleichen Window differenzierbar bleibt. Zu diesem Zweck wird beim Aufzeichnen von Objekten darauf geachtet, dass nicht-veränderliche Eigenschaften gewählt werden.
Physikalische Eigenschaften wie horizontale/ vertikale Bildschirmposition, Länge, Breite, ... werden ignoriert. Die Namen der GUI-Map-Elemente sind frei wählbar.
Nähere Erklärung: sie Kapitel "GUI Map".
Beim Abspielen eines TestScripts sucht sich WinRunner den, bei einem TSL-Befehl angegebenen, Objektnamen in der GUI-Map.
Das Objekt kann dabei durch die nicht-physikalische Beschreibung gefunden und benutzt werden; auch dann, wenn sich seine Position im Fenster geändert hat.

Benutzeraktionen bei denen nicht mit Objekten der Windowsoberfläche interagiert wird, wie z.B.: Bewegungen der Maus, werden nicht aufgezeichnet!

2) Analog Mode

Dabei werden Tastatureingaben, Mausklicks und sämtliche Mausbewegungen mit der x- und y-Position aufgezeichnet.
In welches Fenster oder auf welches Objekt dabei geklickt wird, ist dem Script egal.
Mausbewegungen werden in sogenannte Tracks abgespeichert. Im Script können derart abgespeicherte Tracks auch an mehreren Stellen wiederverwendet werden.
Dieser Mode sollte er nur verwendet werden, wenn es unbedingt nötig ist -eispielsweise zum Testen einer Zeichenapplikation.

Zwischen den beiden Modes kann sowohl während der Aufnahme als auch dem Abspielen von Scripts jederzeit hin- und hergeschaltet werden.

3.3 Test Prozess mit WinRunner

1) Erzeuger der GUI-Map

Zuerst sollten alle Objekte der Testapplikation in die GUI-Map eingetragen werden, damit beim Test der Zugriff auf sie ermöglicht ist.
Bei der Aufnahme von Tests werden die angesprochenen Objekte automatisch in eine GUI-Map gespeichert. Um jedoch *mehrere* Applikationen testen zu können, reicht dies nicht aus; es ist dazu jeweils ein eigenes GUI-Map-File von Vorteil.

Rapid Test Script:

Damit können systematisch alle Objekte, die auf herkömmlichen Wegen erreichbar sind, in die GUI-Map eingetragen werden. Das Script schreibt alle Menüeinträge in die GUI-Map und probiert - gemäß der Konvention - Menüeinträge, die mit "..." enden, zu öffnen. Grund hierfür ist, dass diese Einträge einen weiteren Dialog öffnen.

In allen geöffneten Windows und Dialogen werden sämtliche auffindbaren Windowelemente in der GUI-Map gespeichert.

2) Schreiben der Tests

Dabei wird die Logik der Test Cases im Script aufgebaut.

Im Gegensatz zu WinBatch verfügt WinRunner über eine Vielzahl von Checkpoints und Checkfunktionen für Windowelemente (GUI-Objekte), Bitmaps und Datenbanken.

Das heißt, es braucht keine Get-Funktion mehr aufgerufen und auch im nächsten Schritt kein Vergleich mit dem erwarteten Ergebnis gemacht werden, denn: man übergibt der Check-Funktion den Wert mit.

Beim Erstellen der Tests werden die erwarteten Ergebnisse in einen "expected result"-Ordner im Testordner abgespeichert.

3) Debug Tests

Wie WinBatch verfügt WinRunner über einen DebugMode, um herauszufinden, ob die Tests so laufen, wie sie sollen.Man kann hierfür Breakpoints setzen, Variablenwerte beobachten, durch das Script steppen, ...

Testergebnisse werden in einem Debug-Verzeichnis abgespeichert.

4) Run Tests

Hierfür wird der Verify Mode verwendet. Jeder Testlauf bekommt einen Namen, unter dem die Ergebnisse abgespeichert werden. Bei jedem Checkpoint wird das aktuelle mit dem gespeicherten ("expected") verglichen und bei einem Unterschied im "actual result"-Verzeichnis (das den gleichen Namen trägt wie der Testlauf) gespeichert.

5) View Result

Anders als WinBatch verfügt WinRunner über ein automatisches Testresult. In diesem werden aller Major-Events, die während des Tests aufgetreten sind (wie Checkpoints, Error Messages, System Messages, ...) automatisch eingetragen.Zusätzlich kann man noch selber Nachrichten und Checks dazuschreiben.Gutgegangene Checks werden grün, unterschiedliche Checks werden rot dargestellt.

3.4 Editor

Der Editor verfügt über Syntax Coloring (Keywords, Strings, Kommentar und restlicher Code werden unterschiedlich eingefärbt) und AutoCompletion; bei Drücken auf Strg+Space wird entweder ein Fenster mit den zu dem jeweiligen angefangenen Wort passenden Funktionen geöffnet oder, wenn es nur mehr eine einzige Funktion gibt, die zu diesem Wortanfang passt, die Funktion vervollständigt. Bei dieser Liste werden alle Funktionen genommen, die im Function Generator definiert sind.

3.4.1 Function Generator:

In ihm sind alle bekannten TSL-Funktionen definiert. Der Function Generator wird bei der AutoCompletion verwendet, kann aber auch als eigener Dialog geöffnet werden.
Die Funktionen werden Gruppen zugefügt, wobei eine Funktion auch in mehreren Gruppen definiert sein kann.
Man kann auch selbst Funktionen in den Funktions Generator einfügen. Dafür stehen einige TSL-Funktionen zur Verfügung. Leider müssen diese Funktionen bei jedem Start des WinRunners neu eingefügt werden.
Dazu gibt es mehrere Möglichkeiten: siehe AutoScriptRun, AutoLibLoad

3.4.2 AutoScriptRun, AutoLibLoad

In den Optionen können eine Default Gui Map und ein StartUp Test eingestellt werden. Diese werden beim Start von WinRunner ausgeführt bzw. geladen. Unabhänig davon gibt es im WinRunner Programm Verzeichnis „dat“ einen TSL_init WinRunner Test der beim Starten von WinRunner immer ausgeführt wird. In dieses Script File können ebenfalls Erweiterungen z. B.: zum Laden von Libraries, geschrieben werden.

3.5 Test Directory Struktur

Ein Test im WinRunner ist im wesentlich ein Verzeichnis mit dem TestNamen. In diesem Verzeichnis steht eine Datei namens script, die das Script enthält.
Erwartete Werte werden in ein exp Subverzeichnis gespeichert. Ergebnisse von Debug Testläufen kommen alle in ein debug Subverzeichnis. Jeder Testlauf ist wiederum ein eines Subverzeichnis, mit dem Namen des Testlaufs.

3.6 TSL

Der Funktionsumfang ist mit dem von WinBatch vergleichbar. Nur; dass diese mehr auf das Testen ausgelegt sind und es deshalb schon Vergleichsfunktionen für eine oder mehrere Objekte und deren Attributten, Bitmaps, Datenbanken, Windows, ... gibt.
Außerdem gibt es für alle erkannten Objektklassen spezifische TSL Statements, wie z.B. für Listen Objekte ein *list_select_item.*

3.6.1 Variablenmanagement

Es gibt wie beim WinBatch keine Datentypen, jedoch wird auch beim WinRunner zwischen numerischen und String Werten unterschieden.
Jedoch gibt es unterschiedliche Scopes. Ohne Angabe wird *auto* verwendet, was einem Test-, bzw. Funktionweiten bekannten Scope gleichkommt. Mittels *public* können Testübergreifende Variablen definiert werden. Solche Variablen können in einem anderen Test mit *extern* wieder abgefragt werden. Static definiert in Tests Varaiblen die nur in diesem Test bekannt sind. In Funktionen wird der Variablenwert bei jedem Funktionsaufruf mitgenommen.

3.6.2 Funktionen

Eine Definition einer Funktion muss folgenden Aufbau besitzen:
Scope function Funktionsname(parameter (bis zu 8)) { code}
Als Scope werden die gleichen wie bei den Variablen benutzt.
Public: Funktion kann auch in anderen Tests aufgerufen werden.
Static: Nur testlokale Funktion, auch in Compiled Modules möglich
Extern: ein externe Funktion,z.B. aus einer DLL (siehe DLL Kapitel) wird geladen
Variablen müssen in Funktionen vor allen anderen Statements deklariert werden.

3.6.3 Test Arten

Es gibt zwei Arten von Tests:

1) Main Tests:
   Diese sind Tests im herkömmlichen Sinn und werden von oben nach unten durchlaufen.
2) Compiled Modules:
   Diese dienen als Ersatz für Libraries. In ihnen werden nur Funktionen definiert und kein Testcode (der nicht in einer Funktion steht). Durch Laden solcher Tests werden die Funktionen aufrufbar.

3.6.4 Synchronisierung

Viele TSL Statements lassen eine Synchronisierung mit dem Testtool zu, ohne dass immer ein Menge von WarteFunktionen aufgerufen werden muss. Dabei wird dem Befehl noch eine Zeit übergeben, wie lange er versuchen soll den Befehl richtig auszuführen. Erst wenn diese Zeit addiert mit der generellen Wartezeit, verstrichen ist, wird ein Fehler ausgegeben bzw.eingetragen.
z.B.
win_exists(„Notepad“,2)
WinRunner wartet 2 Sekunden (+ z.B. 10 generelle Wartezeit, ist einstellbar), ob er das Fenster finden kann.

Solche Synchronisierungen sind praktisch, da bei vielen Aktionen, z:b. File Öffnen, deren Zeitdauer nicht vorausgesagt werden kann, da dies vom aktuellen Systemzustand abhängt.

3.6.5 Texterkennung

Für alle Windowelements, wo WinRunner kein Objekt erkennen konnte, oder auf Attribute nicht zugegriffen werden kann, gibt es noch die Möglichkeit mittels Texterkennung, Texte zu lesen. Hierfür gibt es Funktionen, die relativ auf ein Fenster oder welche, die relativ auf ein erkanntes Objekt die Bildschirmpositionen, wo der Text erkennt werden soll, angeben.

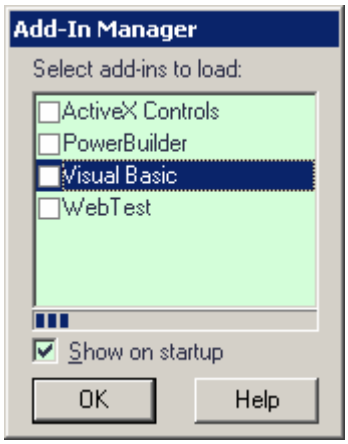
3.6.6 Code Beispiel:

```
public function pfOpenNotepad()
{
extern pTestPath;
invoke_application(pTestPath &
"tools\\kill\\PSKILL\\KillNotepad.bat","",pTestPath &
"tools\\kill\\PSKILL\\",SW_HIDE);
wait(1);
system("notepad.exe");#open notepad
if (win_exists("Notepad",2)!=E_OK){
      tl_step("Open Notepad",FAIL,"Notepad not opened");
      return(E_GENERAL_ERROR);
}
return(E_OK);
}
#-------------------------------------------------------------------------------
-----------
public function pfOpenNewNotepad()
{
system("notepad.exe");#open notepad
if (win_exists("Notepad",2)!=E_OK){
      tl_step("Open Notepad",FAIL,"Notepad not opened");
      return(E_GENERAL_ERROR);
}
return(E_OK);
}
#-------------------------------------------------------------------------------
-----------
public function pfCloseNotepad()
{
auto i=0;
while(win_exists("Notepad")==E_OK){
      wait(1);
      win_close ("Notepad");
      i++;
      if (i>10)
            break;
}
}
```

3.7 Add Ons

Um spezielle Applikationen zu testen, gibt es offizielle Add Ons.
Add Ons müssen beim Start von WinRunner geladen werden. Hierfür kann man in den Optionen einstellen, wie lange man einen entsprechenden Dialog, der alle installierten Add Ons anzeigt, beim Programmstart sehen will. In diesem Dialog kann man mittels Checkbox die Add Ons auswählen, die geladen werden sollen.
In Add Ons sind meist neue TSL-Statements und Tools, die in den WinRunner und Funktion Generator eingebunden werden und genau auf diese Art von Applikationen abgestimmt sind.
Beispiele für solche Add Ons sind:

- ActiveX Controls
- PowerBuilder
- VisualBasic
- WebTest
- Java

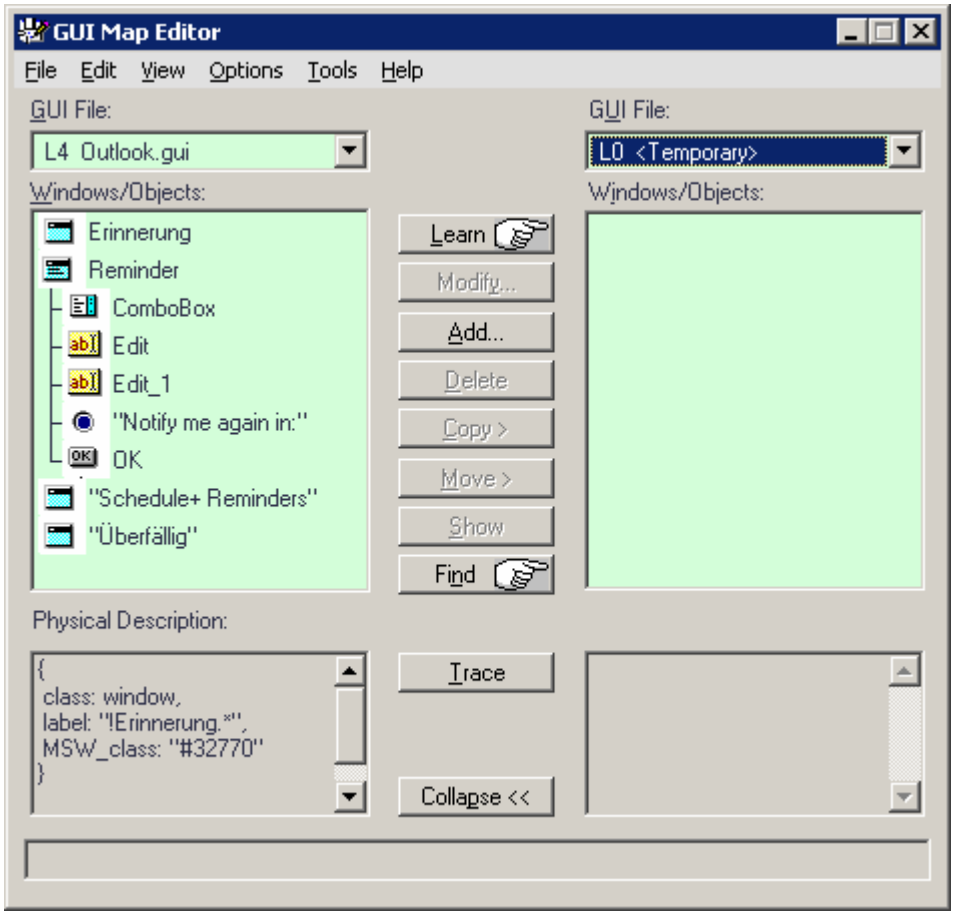


Add Ons verändern teilweise auch bestenden Tools.
z.B. muss Java Add Oninstalliert und geladen sein, damit der GUI Spy Java Objekte erkennt.

3.8 GUI Map

Die GUI Map ist ein Mapping zwischen logischen Namen und Fenstern und Objekten aufgrund ihrer eindeutig voneinanderunterscheidbaren Attribute.
Dadurch ist es leicht möglich die Tests zu warten, wenn sich etwas im GUI der Testappliktion ändert, wie z.B. Der Windows Title oder ein verschobender Button.
Die GUI Map wird in Files gespeichert und kann mittels den TSL Befehl *load_gui* geladen werden.
Es gibt einen Dialog mit dem sich die GUI Map und deren gelade Files angesehen werden kann.
Dies ist auch notwendig da es im speziallfall nötig ist die gelernten Attribute der Objekte und Windows zu ändern. Dabei gibt es auch die Möglichkeit Wildcards einzusetzen.
Objekte werden als Childs von Windows gesehen. Dies ist auch sinnvoll da es unter Windows keine Objekte ohne Fenster gibt (auch Taskbar und Start Menu sind Fenster!).
Außerdem muss beim Zugriff auf Objekte mittels TSL Fuktion, immer zuerst der Fokus auf dessen Fenster gesetzt werden.
Das class Attrbute ist das Hauptattribute das WinRunner für die Identifikation benutzt. WinRunner kennt folgende Class standardmäßig:

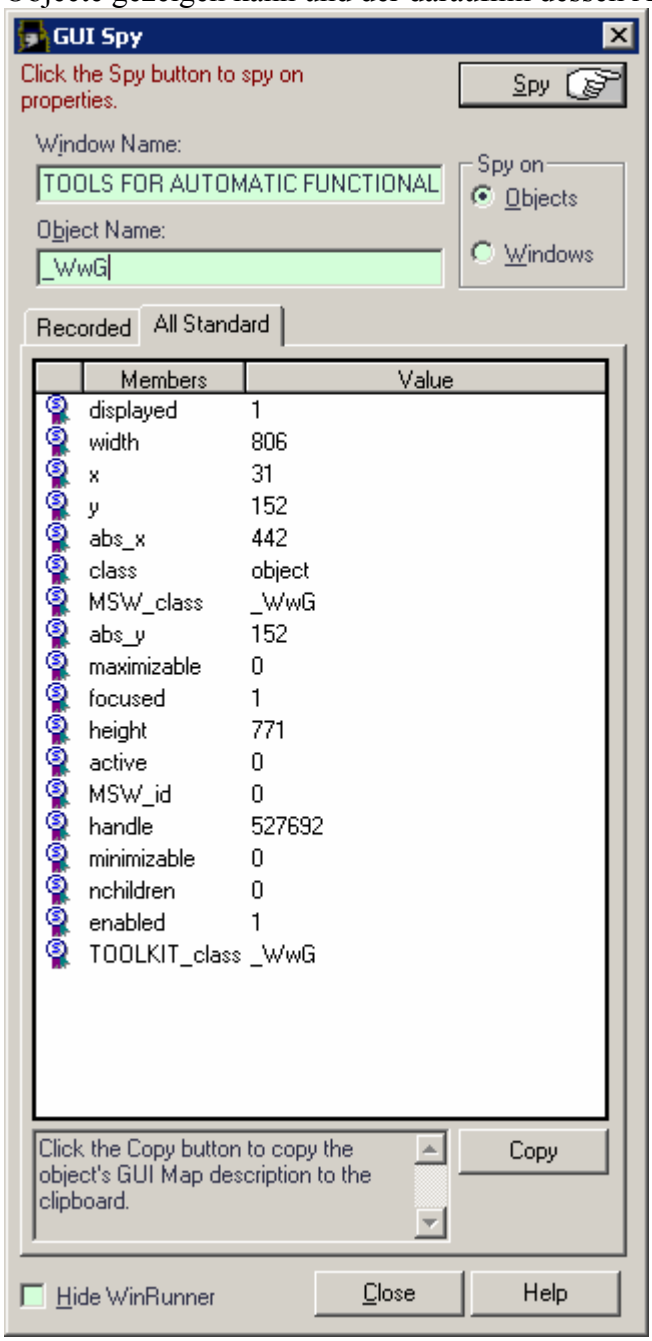
- calendar
- check_button
- edit
- frame_mdiclient
- list
- menu_item
- mdiclient
- mic_if_win
- object
- push_button
- radio_button
- scroll
- spin
- static_text
- status bar
- tab
- toolbar
- window



Für die meisten dieser Klassen gibt es eigene TSL Funktionen mit darauf abgestimmter Funktionalität

3.8.1 GUI Spy

Um die Attribute von GUI Map Einträgen ändern zu können ist es nötig zu wissen welche Attributte das Objekt/Fensterhat und welche Werte sie haben. Dazu gibt es einen GUI Spy,mit dem auf Objecte gezeigen kann und der daraufhin dessen Attribute visualisiert.



3.9 Testlaufarten

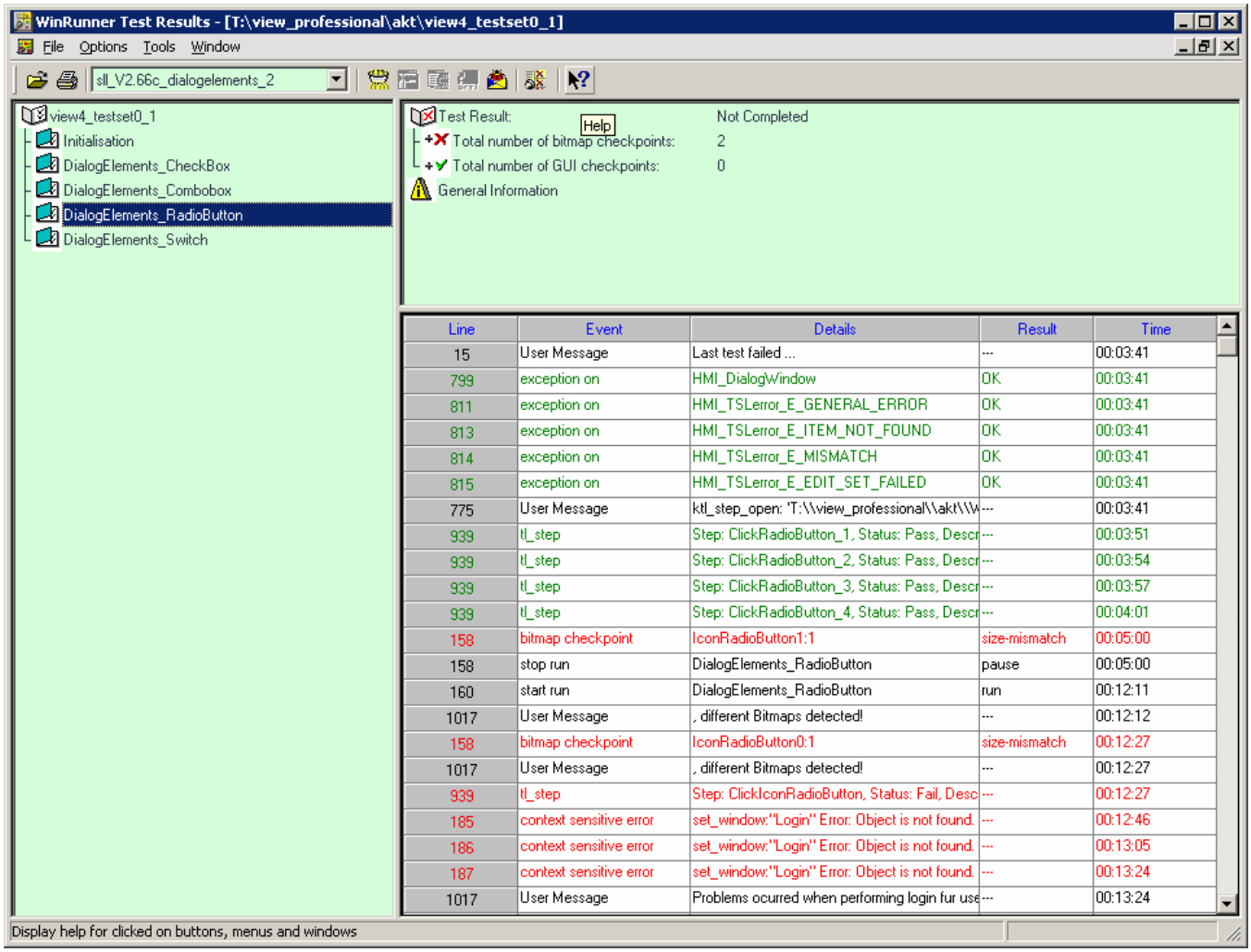
Der Test kann in folgenden Modes gelaufen werden.

- Update: Alle GUI Checkpoints, Bitmaps,.. im expectedVerzeichnis werden aktuallisert.
- Debug: Alle Daten werden ins Debug Verzeichnis geschrieben
- Verify: Normaler Testlauf; es wird ein eigenes Verzeichnis für jeden neu benennten Testlauf erstellt

Außerdem kann ein Batch Modus eingeschalten werden. Dieser ist vergleichbar mit den Error Modes von WinBatch. Wenn er eingeschaltet wird, werden keine GUI Checkpoint und Bitmap Missmatches mehr als Dialog dem Benutzer gezeigt, sondern sofort als Fehler ins TestResult eingetragen.

3.10 Test Result

WinRunner schreibt alle wichtigen Meldungen, wie Systemmeldungen Fehlermeldungen und Usermeldungen in das TestResult. Checkpoints und Checkfunktionen werden je nach State (Test war ok oder not ok) verschiedenfärbig dargestellt. Es gibt auch Funktion denen Expected und Actuall Werte selber übergibt und die ebenfalls einen solchen Eintrag im Test Result generierieren. User können außerdem reine Meldungen ins Result schreiben. Das Result kann gefiltert werden. Es besteht aus einem Testtree, der die Struktur anzeigt, wie die Tests aufgerufen worden sind (nur Main Tests). Einer Auswahlbox, von welchem Testlauf das Rsult angezeigt werden soll. Einem Test Log, das für jeden Test die Erwähnten Ausgaben vom Script enthält. Und einer Test Summery, die den Status des gesamten Tests angibt, wobei ein Test nur dann gut gegangen ist wenn kein Punkt FAILED ist.



3.11 DLL Imports

AuchimWinRunner können Funktion aus DLLs verwenden werden. Dazu muss die DLL mittels load_dll geladen werden und die Funktionen als extern deklariert werden.

4 Verschiedene funktionelle Testtools im Vergleich:

Applikationen unterscheiden sich in vielerlei Hinsicht. Ein Unterschied liegt im Betriebssystem, auf dem sie laufen sollen: die größten Gruppen sind hier Unix/Linux- und Microsoft-Betriebssysteme. Es existieren allerdings auch andere Systeme, für die Software entwickelt und damit Testsoftware benötigt wird.
Ein weiterer Unterscheidungspunkt ist das GUI. Zwei große Gruppen sind die Webanwendungen und die Anwendungen mit externer Visualisierung. Wichtig ist außerdem die Technologie, mit der die GUI erstellt wurde. Dies ist relevant, damit das Testtool auch die Elemente dieser Technologie ansprechen kann. Beispielsweise muss beim Zugriff auf eine CheckBox in einem Panel einer Java-Applikation das Testtool selbst Java unterstützen; da ansonsten nur Mouseclicks und Tastatureingaben, aber keine Objektzugriffe möglich sind.
Für diese Bereiche ein einheitliches Testtool zu generieren, das alle Anforderungen optimal erfüllt, ist fast nicht möglich. Daher haben sich die meisten Hersteller von Automatic-Functional-Testing-Tools auf eine Gruppe spezialisiert. Noch dazu sind die Personen, die die Software einsetzen wollen sehr unterschiedlich, wodurch sich zusätzlich noch die Differenzierung in entwickler- und in anwenderspezifische Tests ergibt.
Viele Tool-Hersteller werben bei ihren Produkten vor allem mit der einfachen Einsetzbarkeit, mit einem Minimalmaß an notwendigen Vorkenntnissen und schneller Erstellung von Testcases. Die intuitivste Methode zum Erstellen von Tests ist daher auch die am häufigsten umgesetzte: Record&Replay (Aufnahme&Wiedergabe) ist ein einfacher Weg, um einen Testfall, den man manuell durchführt, im Nachhinein nochmals zu wiederholen. Um nun diese Aufzeichnung verändern zu können, greifen viele Hersteller in ihren Tools auf Scripts zurück: dabei wird das Aufgezeichnete in ein Script umgewandelt und kann dann editiert werden. Somit entstehen neue Testfälle, die nachher ausgeführt werden können. Da auf diese Weise sehr viele Testfälle entstehen können, besitzen viele Tools zusätzlich ein Testmanagement, das die einzelnen Testcases organisiert und verwaltet. Die folgenden Punkte geben einen Überblick über verschiedene Testtools und deren Einsatzgebiete:

4.1 .Test

Dieses Tool dient vor allem dem Testen von .NET Applikationen, also C#, VB.NET und MC++. Dabei werden Unit-Tests durchgeführt. Das Tool lässt sich in die Visual Studio Umgebung integrieren und führt die Tests entweder geführt oder aber eigenständig durch. Während der eigentliche Test abläuft, kann auf dem System auch weitergearbeitet werden. Als zusätzliches Feature verfügt das Tool über einen Coding Standard Analysator, der den Source Code gemäß der .NET Framework Design Guidelines überprüft.

4.2 Auto Tester One

Dabei handelt es sich um ein sehr einfaches Tool, das ohne größeren Einlernaufwand bereits die Durchführung einfacher Tests ermöglicht. Das Testen von Webapplikationen ist damit genauso möglich wie das Testen von Client-Serverapplikationen. Dabei verfolgt das Programm zwei verschiedene Schienen, von denen eine für den Einsteiger ist – und die meisten Funktionen mittels besonders umfangreicher Aufnahme/Wiedergabe ermöglicht und die andere für den Experten. Für letzteren gibt es ein umfangreiches Scripting-Konzept. Alle Testfälle können in ein Management verpackt werden. Das System setzt keine Programmiergrundkenntnisse voraus.

4.3 Bug Huntress

Dieses System wurde speziell für das Testen von Palm-Applikationen entwickelt. Dafür braucht man in diesem Fall nicht einmal die dazupassende Hardware, sondern für den Test wird die

Hardware emuliert. Das System ist für verschiedene Palm-OS geeignet und verwendet Blackbox-Testing. Auch dieses System setzt keine Programmiergrundkenntnisse voraus. Ein Bildvergleich des Screens mit gespeicherten Bildern zum Erkennen des richtigen Inhaltes ist möglich.

4.4 CAPBAK/X, CAPBAK/MSW

Dieses Tool, das es einerseits für X-Windows aber auch für Microsoft Windows gibt, ermöglicht automatisiertes Testen und Organisieren der Testfälle. Auch dieses Tool hat einen eingebauten Bildvergleich – zusätzlich verfügt es aber noch über eine Erkennung des Textes eines Bildes mittels OCR. Mit dem Zusatz TestCoverage können ungetestete Bereiche herausgefunden werden. Vor allem in der Erkennung dieser Lücken des Tests bzw. des getesteten Sourcecodes kommt die Stärke dieses Tools heraus.

4.5 Certify

Dieses Tool setzt auf anderen Tools auf und ermöglicht unerfahrenen Benutzern das Testen von Software, allerdings nicht wie viele andere Hersteller via Script oder Aufnahme der Aktionen.
Es nutzt andere Testtools, um plattformunabhängig seine Tests durchführen zu können. Einsetzbar sind z.B. WinRunner, RationalRobot, Silktest, ...

4.6 Citra Test

Dient zum Testen von jeglichen Windows-Applikationen. Das Programm vergleicht Images mit abgespeicherten bzgl. korrekter Results und ist in VBA programmierbar. Das Programm bildet Tastatur und Mauseingaben nach und soll damit einen virtuellen User simulieren.

4.7 Code Testing Tools Pro

Dieses Tool ermöglicht mehrere Tests – auch Stresstests, Multithreaded-Tests, Regressionstests, Stabilitätstests,....
Neben GUI-Tests können auch Libraries und DLLs getestet werden. Das Tool verfügt über eine übersichtliche Oberfläche, die auch verschiedenste Statistiken zulässt. Der eigentliche Test ist per Monitoring überwachbar.

4.8 Eggplant

Dieses System benötigt zwei Computer, wobei einer ein MAC ist. Der MAC übernimmt die Steuerung des anderen Netzwerk-PCs. Damit kann das Tool jeden PC testen, der über eine IP-Adresse verfügt und im Netz hängt. Die eigentliche Steuerung erfolgt dann über einen VNC-Service – das System arbeitet damit wie ein virtueller User. Ein großer Vorteil zu anderen Systemen ist damit die geringe Einnahme von Resourcen und das damit wenig verfälschte Testergebnis. Das System arbeitet komplett plattformunabhängig, benutzt einen Bildvergleich zum Verifizieren der Ausgaben und kann vor allem die Interaktion von mehreren Programmen auf dem Zielsystem genau beobachten. Vom kompletten Testfall kann ein Quicktime-Movie erstellt werden, das dem Entwickler einen Fehler veranschaulichen soll.

4.9 Eventcoder

Auch der Eventcoder zeichnet Tests auf und gibt diese wieder. Ein Unterschied ist allerdings, dass er die einzelnen Fenster, die bei der Aufzeichnung angesprochen wurden, genau in der gleichen Reihenfolge auch wieder mit Inputs beschicken kann - damit werden diese Fehler auf Wunsch ausgeschaltet. Auch der Eventcoder baut für weitere Automatisierung auf dem VB-Script auf. Diese Test-Scripts können auch zeitversetzt abgespielt werden. Außerdem verwendet das Programm viele weitere praktische Tools, die gewisse Systemeigenschaften auslesen können (Spy-Tools).

4.10 GUITAR

Dabei handelt es sich um ein freies Framework zum Testen der GUI einer Applikation. Dieses Test-Framework besteht aus mehreren Teilen, die dazu dienen, Testfälle zu erzeugen, auszuführen, die Korrektheit zu überprüfen und Reports zu erstellen. Diese Tools sind der GUI-Ripper, der die GUI der Applikation untersucht und auf einen sogenannten Integration Tree abbildet. Dabei bildet jeder Knoten dieses Baumes eine GUI-Komponente. Weiters zählt der Event Flow Graph Generator, der ein Zusammenspiel aller möglichen GUI-Komponenten in einem Graphen darstellt, zu den eben erwähnten Tools. Der entstandene Graph kann dann editiert werden. Außerdem beinhaltet GUITAR noch einen Testcase-Generator, dieser erstellt Testcases – entweder strukturiert oder zufällig; die Testfälle können aber auch manuell eingegeben werden. Aus diesen wird dann der Coverage-Report erstellt, der die Abdeckung des Tests beinhaltet.

4.11 LabitStudio

Mit diesem Tool ist es möglich, Programmexceptions ausfindig zu machen und deren Gründe zu finden. Das Tool funktioniert für 32-bit Microsoft Betriebssysteme und unterstützt diverse Entwicklertools – dabei findet es heraus, warum eine Applikation auf einem gewissen Computer abgestürzt ist, was genau passiert ist und was der Status der Applikation war, als die Exception passierte. Weiters gibt das Tool den Applikationsstack aus und welche Module der Applikation gerade aktiv waren. Außerdem zeigt es die Umgebung, in der die Applikation eingesetzt wurde.

4.12 MITS.GUI

Dieses Tool dient zur Evaluierung von GUIs und arbeitet dabei ohne jegliche Scripts. Es navigiert selbstständig durch die GUI und verwendet dabei Werte, die der Tester in eine Tabelle eingetragen hat. Ein großer Vorteil liegt in der einfachen Erweiterbarkeit der Tests, wenn ein neues GUI-Element dazukommt. Die Änderungen müssen nur an einer Stelle definiert werden, die Tests laufen dann alle wieder wie vorher.

4.13 PyUnit

Dieses Open-Source-Framework ist das Pendant zu JUNIT mit der Programmier- bzw. Scriptsprache Python. Das Unit Testing-Tool ermöglicht ein einfaches Schreiben von Tests.

4.14 QACenter

Mit diesem Tool sind neben funktionalen Tests auch Load-Tests machbar. Das Tool nimmt Testfälle auf und kann diese in verschiedenen Instanzen wiedergeben, um so auch eine Auslastung zu messen. Die mit veränderlichen Werten startbaren Tests können von Analyseassistenten ausgewertet werden. Die Tests sind mittels Script editierbar. Alle Daten, auch die Antwortzeiten, sind statistisch auswertbar – vor allem Schwachstellen der Applikationen sind so schnell identifizierbar.

4.15 QES/Architect

Dieses System verwendet statt der Scripts Process-Models und speichert alle Daten in einer Datenbank. Das Tool unterstützt alle Systeme mit Textumgebung und vergleicht den Output automatisch. Testfälle können - für Prototyping - generiert werden, aber auch ganz simpel manuell erstellt werden. In einem Testcase-Viewer werden die Modelle mit Inputs und Outputs dargestellt und verwaltet, wobei die eigentliche Wartung und das Management der Testfälle großteils automatisiert erfolgt.

4.16 Repro

Dieses Tool dient dem Aufnehmen von Testfällen – dabei wird ein Video der Testfälle erstellt, das dann dem Entwickler gezeigt werden kann. Das System verlässt sich dabei auf das manuelle Testen,

zeichnet aber alle zuvor ausgewählten Teile auf und gibt diese dann zeitgleich auch wieder. Damit sieht der Entwickler das Zusammenspiel der einzelnen Komponenten. Dabei wird auch die Hardware bzw. die Umgebung in der die Software läuft bedacht.

4.17 Silktest

Dieses Tool ermöglicht das Testen von Webapplikationen mit allen gängigen Browsern. Ein Object-Browser ermöglicht das Ansehen aller Klassen mit deren Methoden und Eigenschaften. Auch klassische Client-Server Applikationen können getestet werden.

4.18 Smalltalk Test Mentor

Man unterscheidet den Test-Mentor für VisualAge und jenen für VisualWorks. Beide Tools dienen dem Testen von Smalltalk Applikationen und bieten den Testern auch die Möglichkeit, ihre Tests per Smalltalk zu erweitern. Das Tool greift dabei direkt auf beliebige Klassen des Smalltalk Images zu – egal ob es sich um eine Interface Klasse oder eine Model Klasse handelt. Es hat damit vollen Zugriff auf sämtliche Instanzen der User Interface Klassen. Der Test-Mentor wird direkt in die Entwicklungsumgebung integriert, die Testfälle werden als Klassen abgespeichert. Die zwei Variationen - „Runner“ und „Developer“ - dienen dabei den unterschiedlichen Usern – „Runner“ gibt den Überblick und „Developer“ ist für die Unterstützung des Entwicklers gedacht.

4.19 Squish

Dabei handelt es sich um ein Test-Framework für Qt/C++ Applikationen. Es kann die Testfälle aufzeichnen, diese per Scripts verschiedener Sprachen verändern und dann flexibel wiedergeben. Auch das komplett manuelle Schreiben von Testfällen ist möglich. Das Tool kann in verschiedenste Testumgebungen integriert werden. Der TestRunner kommuniziert via Socket Connection mit der laufenden Applikation – dadurch kann eine Trennung zwischen der Testapplikation und der eigentlichen Applikation gewährleistet werden. Beim Aufzeichnen von Testfällen werden keine absoluten Positionen, sondern Clicks verwendet und auch beim Vergleichen der Ergebnisse werden keine Screenshots verglichen, sondern das Ergebnis wird unabhängig von seinen Fonts und Positionen verglichen.
Die Testfälle können auch mittels Debug durchgegangen werden – damit ist ein Testscript auch bei einer bestimmten Stelle anhaltbar. Das Tool verfügt weiters über eine GUI Object Identification. Dabei müssen die Objekte nicht gelernt werden, sondern die Erkennung erfolgt während des Einsatzes. Die Tests können auch in Tabellenform eingegeben werden – dabei werden verschiedene Formate, wie z.B. .csv unterstützt.

4.20 TALC2000

Dieses Tool dient dem Testen von zeichenbasierten Systemen auf Mainframes oder Midrange-Systemen. Dabei wird der Test auch aufgezeichnet und dann wiedergegeben. Der Zugriff erfolgt über einen PC Terminal Emulator. Dabei ist das Tool plattformunabhängig und kann unter verschiedenen Windows – Plattformen verwendet werden. Der Fokus des Programmes liegt dabei auf einer sehr kurzen Einarbeitungszeit und den geringen erforderlichen Vorkenntnissen.

4.21 TestComplete

Dieses Tool vereint den funktionalen Test mit http load Tests, Stress Test und Scalability Test. Ein verteiltes Testen sowie Unit-Testing ist möglich. Das Tool funktioniert sowohl für Win32 sowie .NET als auch für Java- sowie andere Applikationen und ist geeignet zum Testen von Webapplikationen. Dabei verwendet es das HTML Object Model. Die verwendete Scriptsprache ähnelt VB und Javascript, es können aber auch nicht-nacheditierte Tests verwendet werden. Ein durchgehendes Log-Verzeichnis zeigt die Fortschritte innerhalb der verschiedensten Tests – dieses kann als XML- oder HTML-File exportiert werden. Ein Object Browser gibt zusätzlich die

Möglichkeit, sämtliche Objekte und ihre Details zu analysieren – diese Snapshots können auch gespeichert und mit anderen verglichen werden.
Ein zusätzlicher Event Handler ermöglicht das Überwachen von ausgelösten Events. Das gesamte Tool ist COM-basierend und daher relativ sprachunabhängig und ist offen für externe Programme.

4.21.1 Praktische Erfahrung:

Bei der Scriptsprache kann ausgewählt werden, ob man VBScript, DelphiScript, JScript, C++Script oder C#Script verwendet. Statt GUI Checkpoints - wie bei WinRunner - gibt es eine Funktion, die die Objekte eines selektierten Objektes und deren Properties in ein XML-File schreibt. Dieses XML-File kann mittels Check-Funktionen beim Testlauf mit den aktuellen Properties verglichen werden. Der Objekt Browser stellt eine ähnliche Funktionalität wie der GUI Spy unter WinRunner dar.
Eine minimale Version der GUI Map liefert das Name-Mapping, bei dem ein Objekt einen Namen zugewiesen bekommt. Jedoch sind Objekte von Fenstern nicht als dessen Childobjekte definiert und müssen dadurch bei einer Änderung des Window-Objektes ebenfalls geändert werden. Außerdem werden nur Objektnamen durch den selbstdefinierten Namen ersetzt und nicht wie beim WinRunner mit einer eindeutigen Beschreibung des Objekts gespeichert.
Als Work Around kann man die GUI Map selbst im Script implementieren.
Bei Java Applikationen muss man beim TestComplete den Klassenpfad anpassen, damit auf die Java Objekte zugegriffen werden kann.

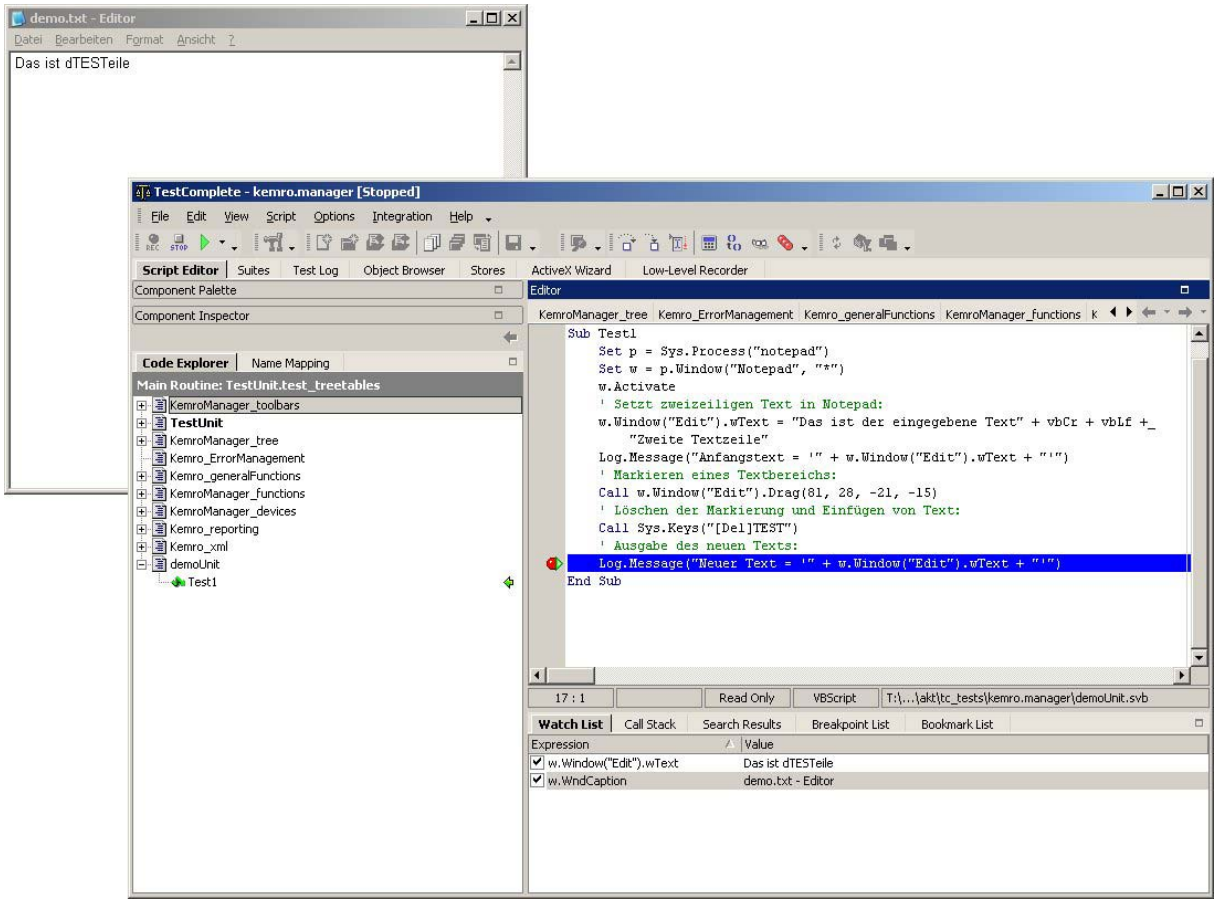
TestComplete hat aufgrund der Fülle von Funktionen, die es anbietet, ein sehr gutes Preis/Leistungsverhältnis.

Beispiel:
Automatisierung von Notepad
Script:

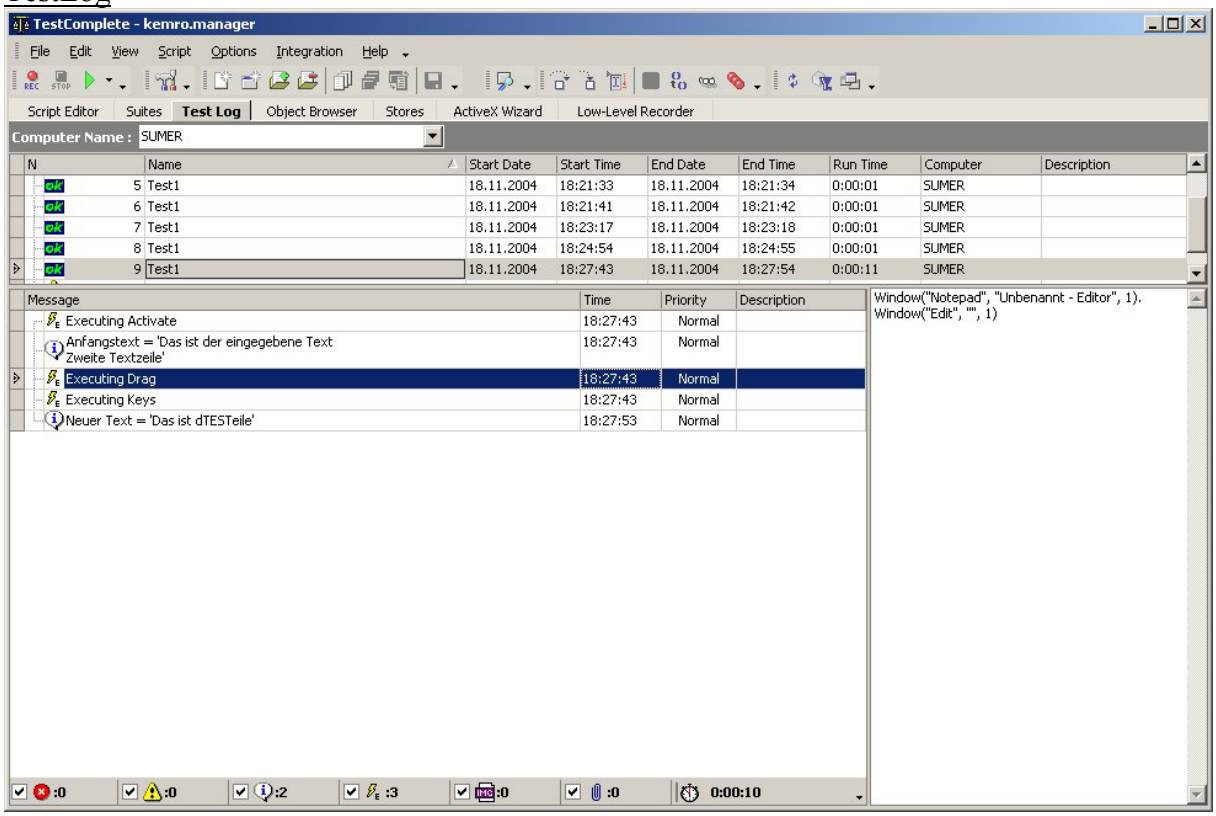
```
Sub Test1
   Set p = Sys.Process("notepad")
   Set w = p.Window("Notepad", "*")
   w.Activate
   ' Setzt zweizeiligen Text in Notepad:
   w.Window("Edit").wText = "Das ist der eingegebene Text" + vbCr + vbLf +_
      "Zweite Textzeile"
   Log.Message("Anfangstext = '" + w.Window("Edit").wText + "'")
   ' Markieren eines Textbereichs:
   Call w.Window("Edit").Drag(81, 28, -21, -15)
   ' Löschen der Markierung und Einfügen von Text:
   Call Sys.Keys("[Del]TEST")
   ' Ausgabe des neuen Texts:
   Log.Message("Neuer Text = '" + w.Window("Edit").wText + "'")


End Sub
```

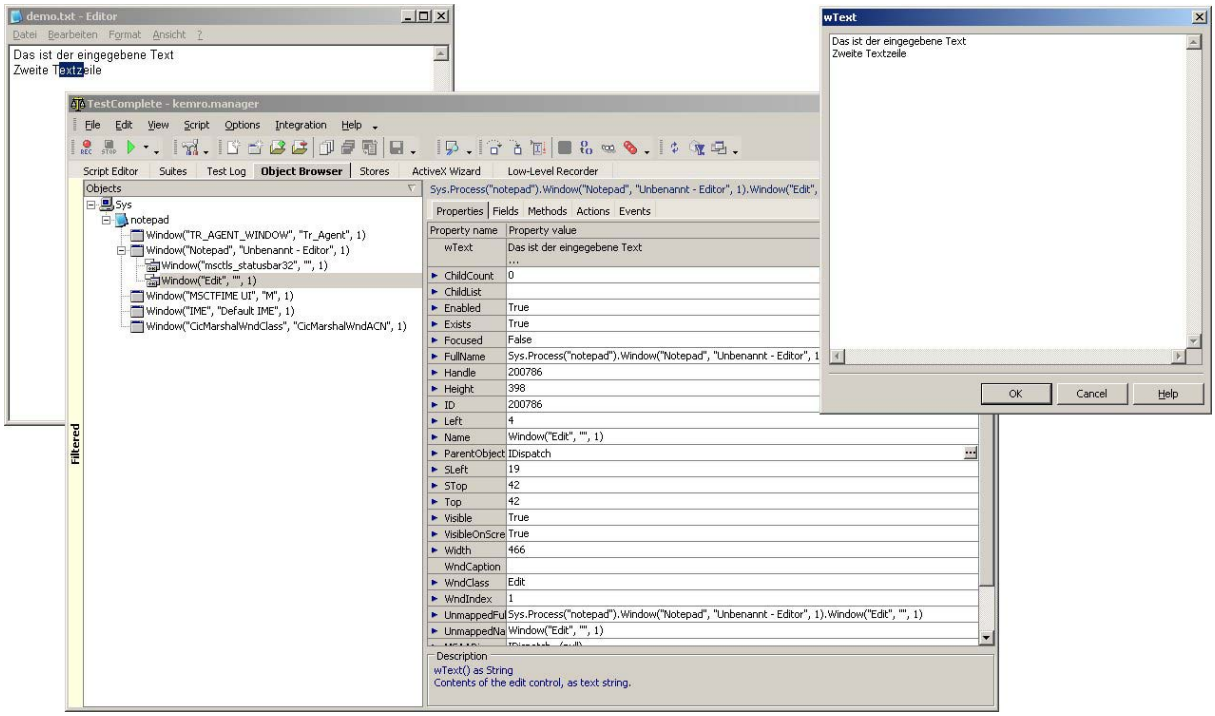
Screenshots:
ScriptEditor:



TestLog



Object Browser:

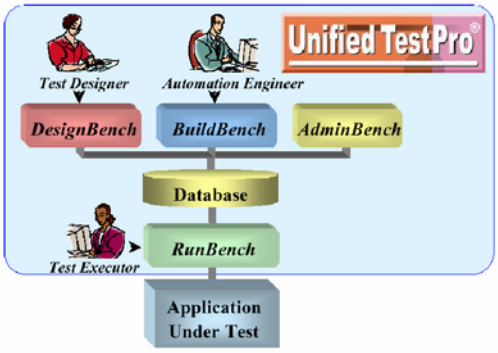


4.22 TestRobot

Dieses Tool emuliert einen humanoiden Tester mit den gleichen Ein- und Ausgabebedingungen . Es verwendet ein objektorientiertes XML-Script für die Tests, wodurch eine leichte Managebarkeit versprochen wird. Das Tool dient zur Überprüfung der Funktionalität als auch der Performance. Das Tool ist plattformunabhängig – es kann auch mehrere unterschiedliche Applikationen gleichzeitig testen – z.B. eine Webapplikation, eine Desktopapplikation und Hardware-Komponenten. Es kann mit anderen Testtools, wie z.B. JUnit zu einer Einheit kombiniert werden.

4.23 Unified Test Pro

Dieses Tool ist spezialisiert auf Keyword Driven Test-Design. Dabei wird das eigentliche Testen in das Design der Test Cases - die von Test Designern erstellt werden - und in die Generierung von Keywords - die von den Automation Engineers (den technischen Experten) implementiert werden, getrennt. Dieses Rollendesign schafft eine Unabhängigkeit vor allem vom technischen Tester und spart somit Zeit. Alle Teile des Tests kommen in eine Datenbank, in die auch automatisierte Daten einfließen. Aus dieser ruft dann der Test Executor via RunBench die Tests auf. Dieses System wird sowohl für das Testen von GUIs bzw. Webapplikationen wie auch für Client/Server Applikationen, API Tests, Telecom/Voice Tests und Mainframe Tests verwendet.



4.24 Vermont High Test Plus

Dieses Tool funktioniert auf Microsoft Plattformen bis hin zu WinXP und unterstützt sämtliche Windows-Objekte. Die Controls der Objekte können gelernt werden. Die eigentliche Generierung der Testfälle erfolgt durch Aufzeichnung und Playback. Das so entstehende Script kann nachher editiert werden. Die Ergebnisfenster können mit dem erwarteten Ergebnis verglichen werden – sowohl als Bildvergleich wie auch als Vergleich der Objekteigenschaften. Ein integrierter Debugger erlaubt das genaue Durchsteppen durch den Test und somit die Fehlereingrenzung. Ein Testmanager ermöglicht die Verwaltung der Testfälle. Das System ist für langwierige Testfälle - die unbetreut ablaufen - ausgelegt.

4.25 WinEZ

Auch dieses Tool ermöglicht das Testen per Aufzeichnung und Playback. Die Ergebnisse werden zerlegt und die Daten der erwarteten GUI ausgewertet – daher ist auch dieses Tool unabhängig vom Style einer GUI. Die Wiedergabe eines Testfalles kann in Zeitlupe und Schritt für Schritt erfolgen. Über das gesamte Ergebnis wird ein Report erstellt.

4.26 XRunner

Der XRunner entspricht dem WinRunner – allerdings ist dieser für Linux und X-Windows gedacht.

4.27 X-Unity

Dieses Tool wurde speziell für den Einsatz von Extreme Programming konzipiert. Dabei soll es Entwicklern von .NET Applikationen helfen, Test Driven Development durchzuführen. Die UNIT Tests eines einzelnen Programmierers können mit dem Tool ausgeführt werden - aber auch Tests von ganzen Gruppen. Dabei ist das Tool entweder mit einer Standalone-GUI zu haben oder es kann in die .NET Entwicklungsumgebung integriert werden. Erweiterungen wie das Web Centric Development Kit ermöglichen zusätzlich das Auslesen von HTML Headern, das Inspizieren von Cookies und vieler anderer Tätigkeiten, die beim Fehlersuchen in Webapplikationen erforderlich sind. Im Fehlerfall ermöglicht das Tool einen RollBack der letzten Version, die ohne Fehler durchgelaufen ist – dieses Feature ist vor allem knapp vor einem Release sehr vorteilhaft. Das Tool arbeitet hier mit Visual Source Save zusammen. Weiters bietet es noch die Möglichkeit, COM-Komponenten sowie DLLs zu testen.

5 Übersicht der Tools:

	Aufnahme und Wiedergabe	Testen von Webapp.	Objekt Mapping	Autom. UNIT Testing	Integr. Test Management
.Test				X	
AETG Web					
Automate! Test Manager					
Automated Test Designer					
Auto Tester One	X	X			X
Bug Huntress					X
CAPBAK/X, CAPBAK/MSW	X	X			X
Certify					X
Citra Test	X	X			
Code Testing Tools Pro		X	X	X	X
Eggplant		X			X
Eventcoder	X	X	X		X
GUITAR		X		X	X
LabitStudio					
MITS.GUI		X			X
PyUnit				X	
QACenter	X	X	X		X
QES/Architect					X
Repro	/x				X
Silktest					
Smalltalk Test Mentor	X		X	X	X
Squish	X	X	X	X	X
TALC2000	X				X
TestComplete	X	X	X	X	X
TestRobot		X		/x	X
Unified Test Pro		X	X		X
Vermont High Test Plus	X	X	X		X
WinEZ	X	X			X
XRunner (siehe WinRunner)	X	X	X	X	
X-Unity				X	X