



AUTOMATED GUI TESTING

Stephen Morris
TESSELLA SUPPORT SERVICES PLC

Issue V1.R1.M1
January 1999



AUTOMATED GUI TESTING

As software becomes increasingly complex, testing has never been more important. Most programs these days are driven by some form of graphical user interface, and the need to test these has led to the development of specialist test tools.

Introduction

Like all products, software must be tested before it is released to the customer. As the complexity of software grows, so does the amount of testing. Since the same tests must often be repeated many times, and the tests themselves can be time-consuming to carry out manually, the appeal of test automation is clear.

Before the advent of computers running modern windowing systems (such as Motif or Microsoft Windows), many applications were command-line driven. The user typed in a command, the system performed some action and then prompted for the next command. Although requiring the user to remember a large number of commands, this form of interacting with the user did have the virtue of making the testing process easier. Testing could be automated by preparing command scripts using a standard text editor, and getting the program to take its input from the script file instead of the screen. Frequently, this required no changes to the program, merely the reassignment of the input stream at the operating system command level.

Virtually all applications nowadays are driven by a graphical user interface (GUI). Although making life easier for the users, automated testing is now more difficult. It is not so easy to reassign the input stream, and the type of input expected by the program is more varied - button clicks, mouse movements, key presses (and combinations thereof). Fortunately, many specialist tools are now available to ease the process of testing GUI-based applications. They often come with a bewildering variety of features, and not all are suitable for every application. This technical supplement presents an overview of some of the capabilities required of such tools, and the issues that must be confronted when choosing one.

GUI Testing Tool Features

Platform Support

The choice of a tool is very much influenced by the development platform. The 32-bit Microsoft Windows operating system (hereafter referred to as Win32) is very much the dominant platform for development, although a significant number

of applications are still being written to run under UNIX using Motif and X-Windows. Motif and Win32, although they have much in common, are two very different systems, and software written for one will not work on the other without significant alteration. Most GUI test tools in the market are aimed at one or other of the platforms, so the target system is perhaps the major influence on the choice of tool.

If software is being developed for a multi-platform market, although it is feasible to use a separate tools for each environment, a tool that works identically on UNIX and Windows has significant advantages. An oft overlooked factor in use of these tools is one of time; for a moderately complicated product, the time needed to become reasonably competent in its use can be significant. Similarly, the time taken to create and maintain automated tests can be a significant cost, and this will obviously be higher if two GUI test tools are in use.

The way that platform-independence is implemented within the software being tested does have an impact on the choice of tool. Broadly speaking, platform-independence can be achieved in one of four ways:

1. Platform-independent GUI library. No native windowing system calls are used in the software, with all GUI components being implemented by use of a third-party product.
2. Emulation software, in which a third-party library provides the API (Application Programming Interface) of one windowing system (e.g. Win32) on another (e.g. Motif).
3. Java.
4. Platform-dependent code, where separate versions are written for the Windows and UNIX environments.

(For a discussion of the issues surrounding platform-independent GUI's, see the Tessella Technical Supplement "Portable GUI Development") It is possible that some tools may not work properly if using methods (1) and (2), as these can sometimes impose, on the underlying system, their own model of how events are delivered to applications. Many GUI test tools rely on the standard model of the platform for their operation, and a non-standard one may interfere with their operation. If deciding to use a GUI test tool in such an environment, it is important to check for compatibility.

One form of platform independent GUI test tool that works regardless of the type

of application architecture, can be used where the test tool is run on a separate machine to that of the application. The two machines are connected, usually via some specialist hardware, and the test machine drives the application under test. As far as the latter is concerned, it cannot tell how it is being run since the system is driven via the same interface and equipment that is normally utilised by users of the system. This form of testing is typically useful for real-time systems, where the presence of the test tool on the target system may interfere with the application under test.

Test Modes

GUI test tools generally run in one of two modes: position-based or object-based. In position-based mode, tests are recorded (and played back) as a set of explicit mouse movements and key or button presses. This is useful for testing applications that draw their own graphics. Usually a graphics screen will not contain other GUI components, and the selection of an item (or other operation) on that screen is determined by the program (rather than the windowing system) based on the position of the event within the window. There are disadvantages though. If the display resolution or screen size changes between the time the test is recorded and the time it is played back, events may be sent to the wrong part of the window, or even to the wrong window. For the same reason, position-based mode is not really suited to testing windows containing standard GUI components. Should the layout of the window be altered - for example by moving or resizing the objects within it - the test may cease to work, even though there is no change to the functionality of the window. Under such circumstances, object-based mode may be more useful.

In object-based mode, the test software sends the event (keystroke or button press) not to a given position in the window, but to a specific object (X-Windows widget or Win32 control). In this way, the positions and sizes of the objects within a window can alter (whether by changes to the program, or because of differences between window managers) without affecting the operation of the test. Similarly, if the function of a window is extended (e.g. by adding a button) but is otherwise unchanged, a test will still work. Although object-based mode tends to make tests more robust, there are still pitfalls. The test tool will test the GUI, but will not check it for correctness. For example, if a button is hidden by some other control, object-based mode will still activate the button even though it would be inaccessible to a user of the system. A visual check of the interface is still necessary.

The best mode for a test depends on the type of test being undertaken, and most GUI test tools allow tests to comprise a mixture of modes.

Synchronization And Timing Considerations

One of the advantages of test automation is that the computer can generally run a test faster than a human being. This is certainly advantageous, but does mean that a GUI test tool must be particularly aware of synchronization issues; for example if pressing a button on one window must be followed by the pressing of a button on a window created as a result of the initial button press, the tool must ensure that the second window is actually present before performing the second button press. Window systems will typically discard events if they are not expected, so in the example given the second button press will be ignored by the application if the second window is not around to receive it. Since the delay in the second window's appearance may well depend upon system loading, and may be longer when the test is run than it was when the test was created, waiting a given amount of time will not always guarantee that the window has appeared. A similar example is where the program under test indicates that a time-consuming operation is in progress by switching the cursor to an hourglass shape. In this case, the test tool must be able to wait for the cursor to switch back to a pointer before resuming input. An application may have many different ways of indicating this information, and a suitable GUI test tool must be able to cope with them.

Although looking for some particular change in the GUI will be the most usual way to handle synchronization, there may be cases where the only way a test will work is for the tool to wait for a given interval. An example of this would be to test how software recovers from an interruption by pressing a "Stop" button after some specified period. For these cases, the ability to tell the tool to pause a given interval before taking some action is needed.

Output Checking

GUI test tools are often used for regression testing, where a test produces some output that is compared to reference data; a difference usually indicates that an error has crept into the software, i.e. the software has regressed. With a GUI-based program, output is typically window snapshots (whether of GUI panels or drawing areas), and it is useful if the test tool can handle them directly. Most test tools are able to do this, using one of two types of comparison method: bitmap and content.

Bitmap comparison is a straight comparison between two images. Although basically a simple comparison of binary data, the utilities that come with GUI test tools are generally able to present the differences in a meaningful way, by highlighting them. Since these may be only a few pixels in an image of over a million, a clear indication of changes is very important.

Content-based comparison on the other hand, ignores the presentation but compares the underlying data itself. An example of this would be the checking of text in a text window; with content-based comparison, it is possible to perform the comparison even if the font in which the text is displayed is different from the reference copy. (This is something quite useful on UNIX systems, where the output to text windows is often in the default font, something that can vary between platforms.) For this, the tool must have some way of getting to the text content, e.g. optical character reading.

Regardless of the comparison method used in a particular check, an often desirable feature of any comparison utility is the ability to “mask-off” sections of objects being compared. Many types of output include details (such as date and time the test was carried out, ID of the user carrying out the test, etc.) that, although important, can vary between tests. If these are not ignored, a number of false test failures can be generated.

Test Scripts

In the preceding sections, the term “test script” has been used. A script is some form of record of actions that can be fed to the GUI testing tool in order for it to replay the test. Although this can be in any format, it is often very useful to be able to edit scripts directly. This requires a form of user-readable (and editable) scripting language.

Within such a language, a desirable feature is the presence of some form of control structure. With this it is possible to create tests that would otherwise be awkward or time-consuming to set up, e.g. repeating a test many times to check for resource leaks, the abandoning of a test once a predetermined number of errors have been encountered, etc. In fact, many tools have such a sophisticated script language that creation of a test script can be a significant programming task in its own right. For this reason, they often ease the job by including an ability to record tests. This is a process whereby the application is used in the

script from scratch, although exclusively using this method to create tests may lead to maintenance problems. It is usually better to combine script recording with test creation via an editor.

Run reference: G27/SM

Object Details		Settings	
Name		Gain	eh
R.A.	23:12.60	Time const	0.1
Dec.	+44.02.00	Sample freq.	0.5

Optics		Miscellaneous	
Filter	K	Date	27/05/2006
FOV	10	Comments	
Chop freq.	18		
Chop throw	75		
Noc freq.	0.05		
Noc throw	200		

OK Cancel Help

Automated testing removes the drudgery of testing GUIs

Test Management

Creating tests with a GUI test tool is only a small part of the testing process. Storing test scripts, running the tests and inspecting results can take a significant amount of effort. Add to this the need to feed the test failures into the bug reporting process, plus the rerunning of tests to verify bug fixes, and it can be seen that management of testing is a significant task in its own right; some form of test management software is essential.

GUI test tools are usually sold in packages complete with management software. The capabilities of the management component vary between products, but most will be able to handle the storage of test scripts, together with scheduling test runs and summarising the results. Some include such items as bug tracking software, with the GUI test tool using it to raise bug reports when tests fail. One very useful component (paradoxically since the emphasis is on automated testing) is the seamless handling of manual tests. Manual tests will always be required, regardless of how much automation is achieved; for example, if the application provides a print function, a manual verification that a document is actually printed, and that what comes out of the printer is what was intended, is always needed. The ability to treat these tests in the same way as the automated tests can significantly simplify management.

Summary

The high-profile millennium bug put the quality of software under a level of scrutiny as never before, and placed testing high on the agenda of software development managers. With most software now driven by graphical user interfaces of such complexity that manual testing is now a time-consuming and costly task, there is an overwhelming case for automation. This article has covered some of the features available in GUI test tools today, and has highlighted how they can be used to advantage in the testing process.

It would be wrong however, to suggest that they are the panacea for all testing troubles. The process of setting up GUI tests can be expensive, both in terms of cost (for the tools) and effort required to create and maintain the tests. It is only worth automating GUI tests if they will be run enough times to cover the expense. The benefits of the tools may not be seen until several software releases after the tools are introduced into the development process. The tools should, therefore, be seen as a long-term investment. In addition, they are not a substitute for good software development practices (such as reviews and code walkthroughs), nor for other types of tests such as memory leak or coverage testing.

Having said this, GUI test tools definitely have their place in the software development armoury and, if used correctly, can provide significant benefit and cost-saving to the whole process of producing software.

Tessella Support Services plc
Creating Software for Science and Engineering

Tessella's services range from feasibility studies, through system design, development, implementation and ongoing support. Our expertise includes:

Data Analysis Software
Data Capture
Simulation Software
Advanced Graphics
Systems Support
Database Applications

Other Technical Supplements available include:

- | | |
|---|--|
| <input type="checkbox"/> Archiving of Electronic Info | <input type="checkbox"/> Object Oriented Programming |
| <input type="checkbox"/> Active Server Pages | <input type="checkbox"/> Pocket PC |
| <input type="checkbox"/> Automated GUI Testing | <input type="checkbox"/> Portable GUI Development |
| <input type="checkbox"/> Bayesian Statistics | <input type="checkbox"/> Printer Technology Guide |
| <input type="checkbox"/> Beowulf Clusters | <input type="checkbox"/> Real Time Systems |
| <input type="checkbox"/> C++ | <input type="checkbox"/> Regression Testing |
| <input type="checkbox"/> Client-Server Technology | <input type="checkbox"/> Security and the Internet |
| <input type="checkbox"/> COM | <input type="checkbox"/> Simulation |
| <input type="checkbox"/> Computational Fluid Dynamics | <input type="checkbox"/> Soft Computing |
| <input type="checkbox"/> Computer Image Processing | <input type="checkbox"/> Software Design Methodologies |
| <input type="checkbox"/> Decision Support Systems | <input type="checkbox"/> Software Development Cycle |
| <input type="checkbox"/> Electronic Data Capture | <input type="checkbox"/> Software Documentation |
| <input type="checkbox"/> Electronic Lab Notebooks | <input type="checkbox"/> Software Portability |
| <input type="checkbox"/> Excel | <input type="checkbox"/> Software Re-engineering |
| <input type="checkbox"/> Extending the Life of Software | <input type="checkbox"/> Software Specification |
| <input type="checkbox"/> Federal Drug Administration | <input type="checkbox"/> SQL |
| <input type="checkbox"/> FORTRAN 90 | <input type="checkbox"/> UNIX Inter-Process Comms |
| <input type="checkbox"/> Grid Computing | <input type="checkbox"/> UNIX Systems Performance |
| <input type="checkbox"/> High Throughput Screening | <input type="checkbox"/> UNIX Workstations |
| <input type="checkbox"/> Instrumentation | <input type="checkbox"/> Visual Basic 6 |
| <input type="checkbox"/> Integrated Lab Systems | <input type="checkbox"/> WAP |
| <input type="checkbox"/> J2EE | <input type="checkbox"/> Web Services |
| <input type="checkbox"/> Java | <input type="checkbox"/> Windows 2000 Services |
| <input type="checkbox"/> Lims | <input type="checkbox"/> XML |
| <input type="checkbox"/> Linux | <input type="checkbox"/> X Windows |
| <input type="checkbox"/> Microsoft Net | |



INVESTOR IN PEOPLE

Tessella Support Services plc

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

E-mail: info@tessella.com Web Address: <http://www.tessella.com>