GUI Testing: Pitfalls and Process

Atif M. Memon, University of Maryland

Graphical user interfaces have become a nearly ubiquitous means of interacting with software systems. The GUI responds to user events, such as mouse movements or menu selections, providing a front end to the underlying application code. The GUI interacts with the underlying code through messages or method calls.

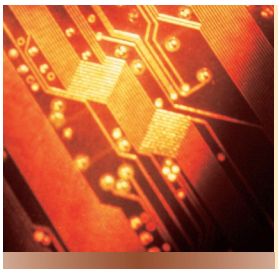
GUIs make software easy to use, and developers are dedicating a larger portion of code to implementing them. GUIs can constitute as much as 60 percent of an application's total code today. The use of GUIs in safety-critical systems is also growing, making their correct operation imperative.

Given their increased importance, testing GUIs for correctness can enhance the entire system's safety, robustness, and usability (http://www.cs.umd.edu/~atif/dissertation.pdf). But GUIs remain a neglected test research area.

TOOLS AND TECHNIQUES

Current GUI testing techniques are incomplete, ad hoc, and largely manual. The most common tools use record-playback techniques. A test designer interacts with the GUI, generating mouse and keyboard events. The tool records the user events, captures the GUI session screens, and then stores the session—usually as a script. The tester later plays back the recorded sessions to re-create the events with different inputs.

This process is extremely labor intensive, often relying on the test designer's ability to generate interesting GUI interactions. An automatic test case generator can provide a higher level of support, but a programmer must code it for all possible decision points in the GUI.



GUIs differ from standard software in ways that complicate traditional testing tools and techniques.

Automated or not, the time-consuming record-playback approach easily misses important GUI decisions. A popular alternative is to release beta copies of the software and let the users do part of the testing. For example, Microsoft tested part of its Windows 95 software by releasing almost 400,000 beta copies.

GUI TESTING PITFALLS

Software testing is already labor and resource intensive—often accounting for 50 to 60 percent of total software development costs—and GUI testing poses further difficulties that traditional software testing techniques do not adequately address.

Coverage criteria

Conventional software uses coverage criteria as a guideline for determining testing adequacy. These criteria define sets of rules that test designers use to determine the type and amount of underlying code to test. For example, a criterion for "event coverage" might require all reachable events in a GUI to execute at least once during a complete cycle of test cases.

However, traditional coverage criteria do not work well for GUIs. First, GUI software differs from the underlying application code in its level of abstraction, so mapping between GUI events and the underlying code is not straightforward. Code-based coverage criteria do not necessarily address problematic interactions between the GUI's user events and the application.

Second, even when experienced test designers focus on specific parts of a GUI, they may still find it impractical to generate all possible test cases for these parts. If the designers have to generate a subset of all possible test cases, they often must select the subset during test case generation. The difficulty of anticipating a test case's fault-detection capability makes it difficult, in turn, to select the most effective subset.

Test oracles

Verifying whether the GUI executes correctly poses a problem. The traditional verification tool is a test oracle—a separate program that generates expected results for a test case and compares them with actual results. In conventional software testing, the tester invokes the oracle after the test case executes and compares the final output with the oracle's expected output.

By contrast, a GUI test case requires interleaving the oracle invocation with the test case execution because an incorrect GUI state can lead to an unexpected screen, which in turn can make further test case execution useless. For exam-

ple, the test case might involve a button on the GUI screen that no longer exists. Thus, a GUI test case should terminate as soon as the oracle detects an error.

Also, if the oracle does not verify the GUI after each execution step in a test case, pinpointing the error's actual cause can become difficult, especially when the final output is correct but the intermediate outputs are incorrect. Consequently, in GUI test case execution, a tester gives the inputs one step at a time and compares the expected output with the GUI's output after each step. This interleaving complicates GUI testing.

Regression testing

Test designers use regression testing to ensure that code modifications do not introduce new errors into already tested code. Regressing the testing process presents special challenges for GUIs.

First, both inputs and outputs to a GUI depend on the layout of graphical elements. Changes in the layout—button placements, menu organization, and so on—can change the input-output mapping and render older test cases useless. Similarly, the expected outputs used by oracles may become obsolete.

Second, developers typically use rapid prototyping for GUIs. This development environment requires efficient regression testing mechanisms that can detect the frequent software modifications and adapt the old test cases to them.

GUI TESTING PROCESS

Addressing the specific pitfalls of GUI testing requires a clear methodology that employs tools and techniques integrated to use a standardized GUI representation. This integration ensures that all test results are compatible with each other.

Other goals include

- task automation to simplify the test designer's work;
- efficient overall test cycle to minimize the frustration of an inherently tedious and expensive process;
- robust testing algorithms to detect each time the GUI enters an unexpected state and to report all information necessary to debug the GUI; and
- portable tools and techniques to allow test information—test cases, oracle results, coverage reports, and error reports—generated on one platform to be used on all other execution platforms.

Finally, the methodology should employ tools and techniques that are general enough to apply to a wide range of GUIs.

GUIs require unique testing techniques, but the process for implementing them is conventional.

Although GUIs differ from conventional software in ways that require unique testing techniques, the overall process for implementing this methodology should follow the steps for conventional software:

- Determine what to test by defining coverage criteria. A GUI coverage criterion might require the execution of each user interface event to determine whether it behaves correctly.
- Generate test case inputs from software specifications and structure. For GUIs, these inputs consist of events such as mouse clicks, menu selections, and object manipulations.
- Generate expected output to compare with actual output. In GUIs, the expected output includes screen snapshots and window positions and titles.
- Execute test cases and verify output. Test cases execute on the software, and the tester compares the output with the expected output from, for example, an oracle.
- Determine whether the GUI was adequately tested. Once all test cases have executed, the tester analyzes the software to check which of its parts were actually tested. In GUIs, the analysis checks events and resulting GUI states.

The last step is especially important in GUI testing, where coverage criteria may not always be available or sufficient.

After testing, the development team corrects any identified problems. The tester then performs regression testing to help ensure the correctness of the software's modified parts and to establish confidence that the changes have not adversely affected previously tested parts.

Test designers develop regression test suites that consist of new test cases and a subset of the original test cases. The subset of the original test cases retests parts of the original software that may have been affected by modifications. The new test cases deal with parts of the software not covered by the subset.

In GUIs, regression testing involves analyzing the changes to the layout of GUI objects.

Individual test designers can instantiate this high-level process to their specific requirements. GUI testing is far from a mature discipline. To help it evolve, both researchers and practitioners must contribute their experience. ■

Atif M. Memon *is as an assistant professor of computer science at the University of Maryland. He is also a scientist at the Fraunhofer Center–Maryland. Contact him at atif@cs.umd.edu.*

Editor: Michael J. Lutz, Rochester Institute of Technology, Department of Computer Science, 102 Lomb Memorial Drive, Rochester NY 14623; mjl@cs.rit.edu