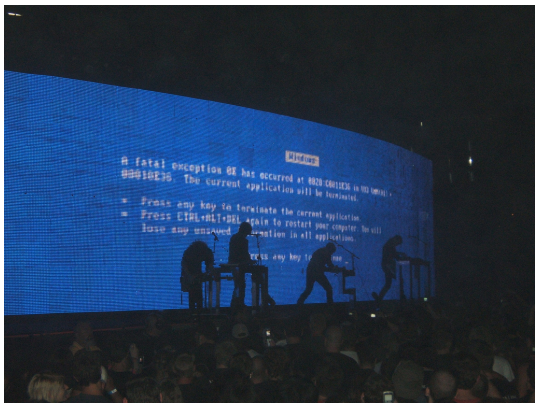
State-of-the-Art in Automated Graphical User Interface Testing

Zebao Gao, Gregory Kramida, Andrew Ruef, and Weiwei Yang
University of Maryland, College Park

People rely on and use software for many tasks, ranging from launching spacecraft, writing reports, and communicating with other people. But, because of bugs, people are not always happy with software. Sometimes, software does not do what it should and this can make users unhappy or sometimes have very serious effects on the world. Software should be a tool that people can rely on, but often it is not.



The (in)famous blue screen of death came to symbolize software errors.

Traditional Software Testing

So how can we ensure reliability of software? For testing software at a low level, there are advanced techniques and tools for software developers to use. However, these tools operate on the software at a much lower level than users of the software. Users typically interact with software via a Graphical User Interface (GUI), and errors there cannot easily be detected by the bug-finding software currently in use by industry or prepared by the research community.

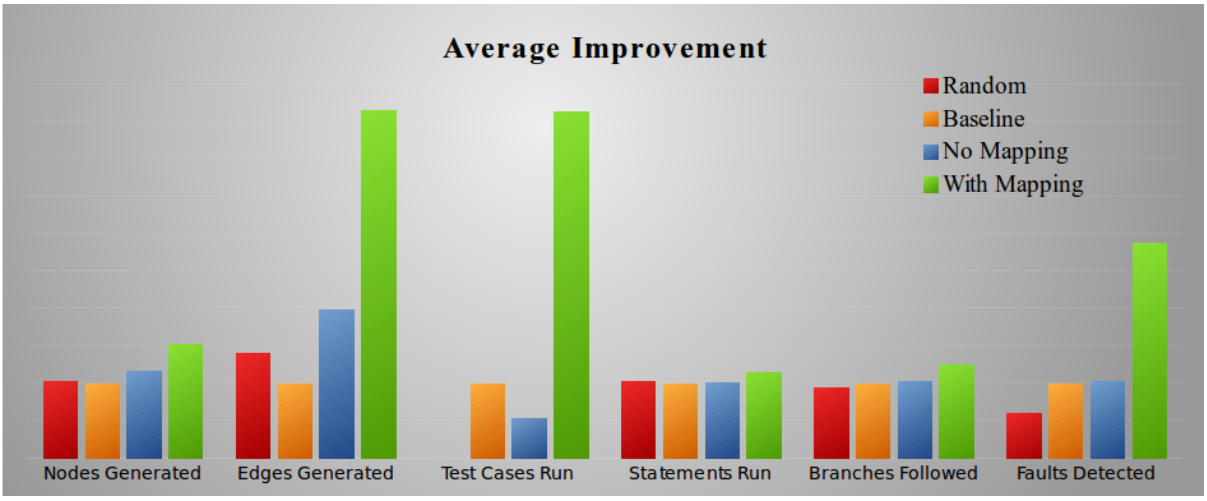
The traditional industry approach is to perform manual testing on graphical applications, which is expensive, time consuming, and often unreliable. This increases the cost of software and the time to market. A study conducted by NIST in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More than a third of this cost could be avoided if better software testing was performed.

Enter GUITAR

How can we improve this? Many testing frameworks either require human testers to manually record/script workflows (e.g., Selenium[1], HP UFT[2]) or randomly trigger controls in the application (e.g. Monkey[3]). GUITAR[4] (GUI Testing FrAmewoRk), designed and developed by Dr. Memon and his research team at University of Maryland, is more systematic and efficient: GUITAR automatically creates an Event Flow Graph, or EFG, to represent a user's interaction with an application.

The graph represents the connections between possible events in the GUI, for example clicking "File" and then "Open". The EFG can then be converted directly and automatically into test cases. And more importantly, the entire process is fully automated.

[1] `http://www.seleniumhq.org/`
[2] `http://www8.hp.com/us/en/software-solutions/unified-functional-automated-testing`
[3] `http://developer.android.com/tools/help/monkey.htm`
[4] `http://sourceforge.net/projects/guitar/`

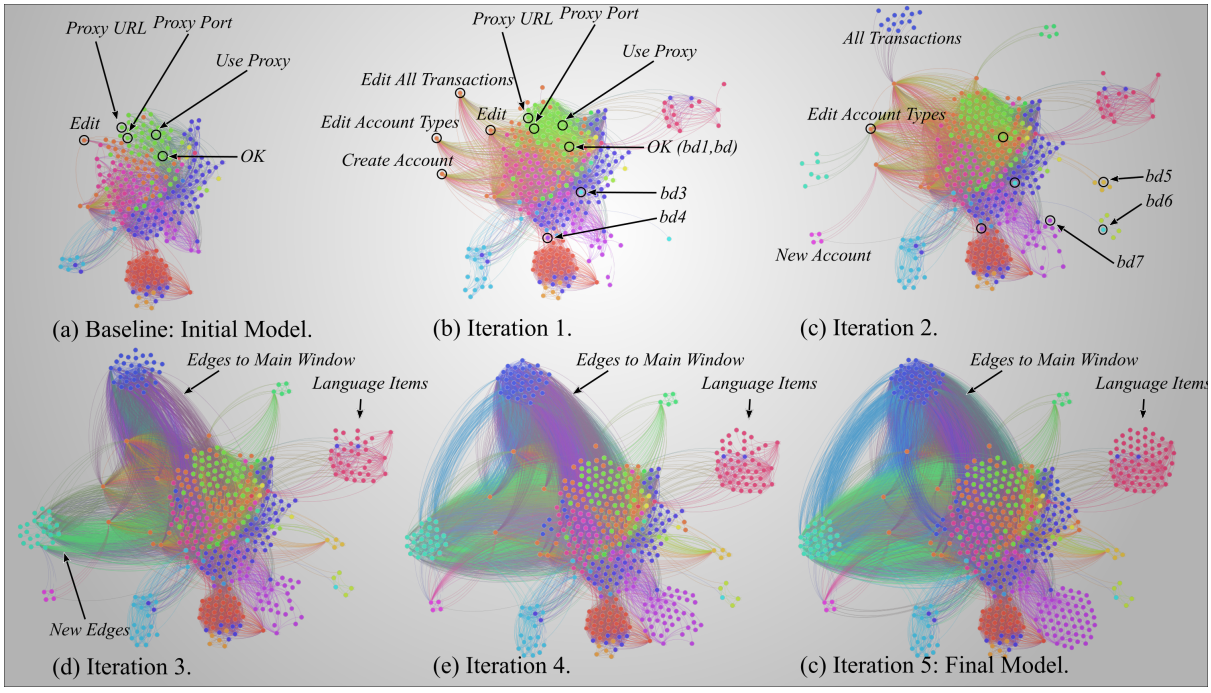


This bar-chart compares the OME* technique with and without context-aware mapping (green and blue bars, respectively) to baseline GUITAR and random path testing.

The OME* paradigm

This is not the end of the story. Software continues to grow in complexity, including interactive complexity. The automatic construction of the EFG is important to the scalability of the testing technique, but the size and complexity of some applications make this difficult. Until recently, it was impossible for GUITAR to build a precise and complete EFG in one run of programs with complex GUIs. Some software is so complex that no state-of-the-art planning techniques are able to depict a full picture of their EFG due to lack of domain knowledge.

Researchers in Dr. Memons lab working on GUITAR have defined a new technique, Observe-Model-Exercise* (OME*). They noticed that one major limitation in GUITAR is lack of context information during execution time. By keeping a context-aware mapping table, this problem is greatly alleviated. In addition, instead of conceptually separating the EFG modeling and test case execution procedures, they introduce a ciclic workflow which observes new widgets and events during execution of test cases.



The event space for a budgeting program Buddi grows from the small model obtained by GUITAR without OME* (a), to an expansive network after five iterations with OME*.

References

[1] Nguyen, B. N., and Memon, A. M. (2014). An Observe-Model-Exercise* Paradigm to Test Event-Driven Systems with Undetermined Input Spaces. *Software Engineering, IEEE Transactions on*, 40(3), 216-234.