

Controlling Factors

ICSE 2013 Tutorial

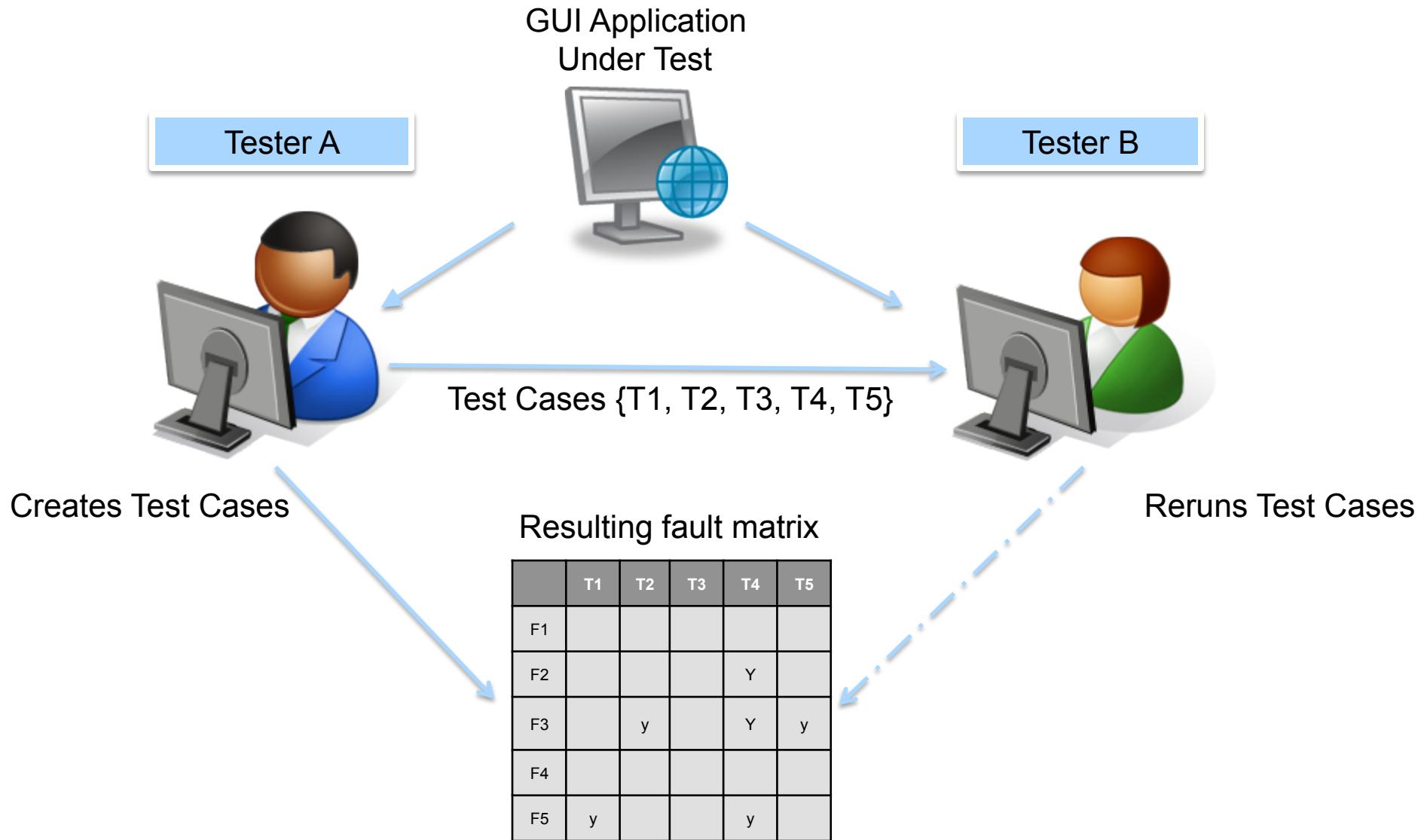
Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness



Atif M. Memon and Myra B. Cohen

UNIVERSITY OF
Nebraska
Lincoln

A Tale of Two Testers



Resulting Fault Matrices

Tester A

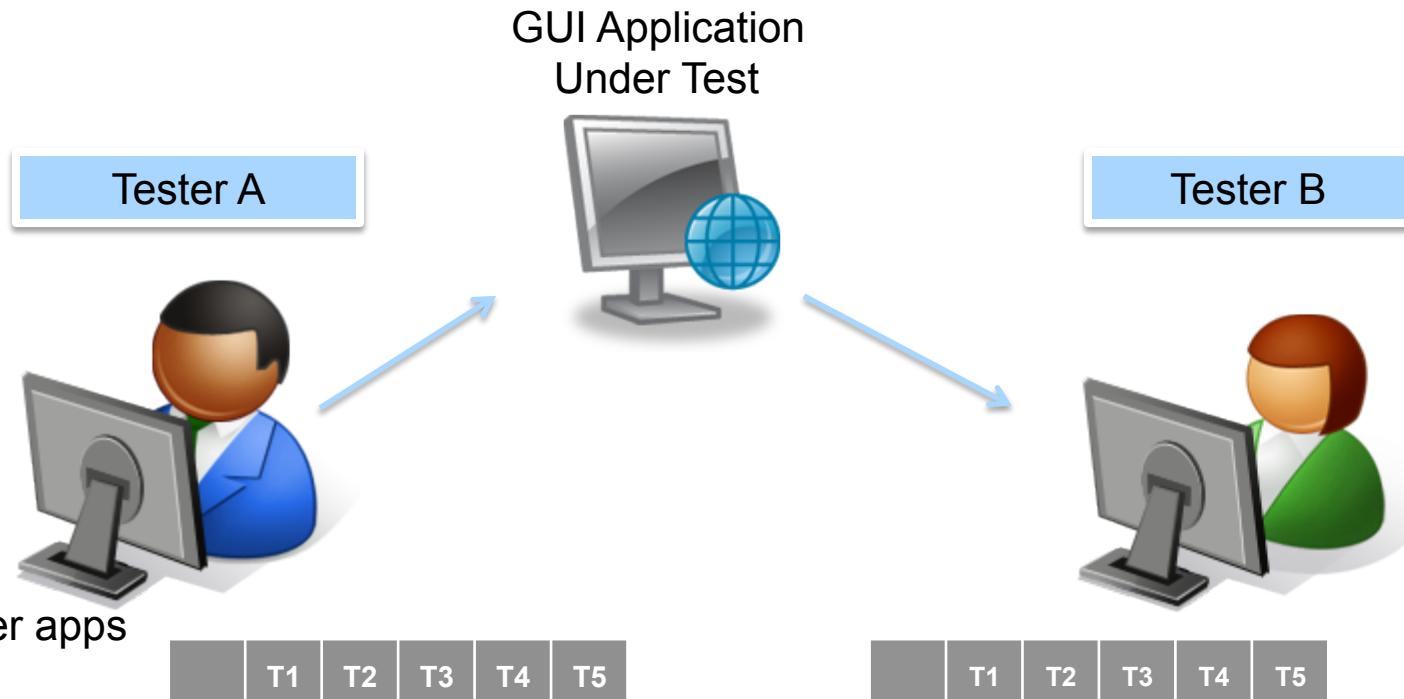
| | T1 | T2 | T3 | T4 | T5 |
|----|----|----|----|----|----|
| F1 | | | | | |
| F2 | | | | Y | |
| F3 | Y | | Y | Y | |
| F4 | | | | | |
| F5 | Y | | | Y | |

Tester B

| | T1 | T2 | T3 | T4 | T5 |
|----|----|----|----|----|----|
| F1 | | | Y | | |
| F2 | | | | | Y |
| F3 | Y | | | Y | |
| F4 | | Y | Y | Y | |
| F5 | Y | | | | Y |

Why the mismatch?

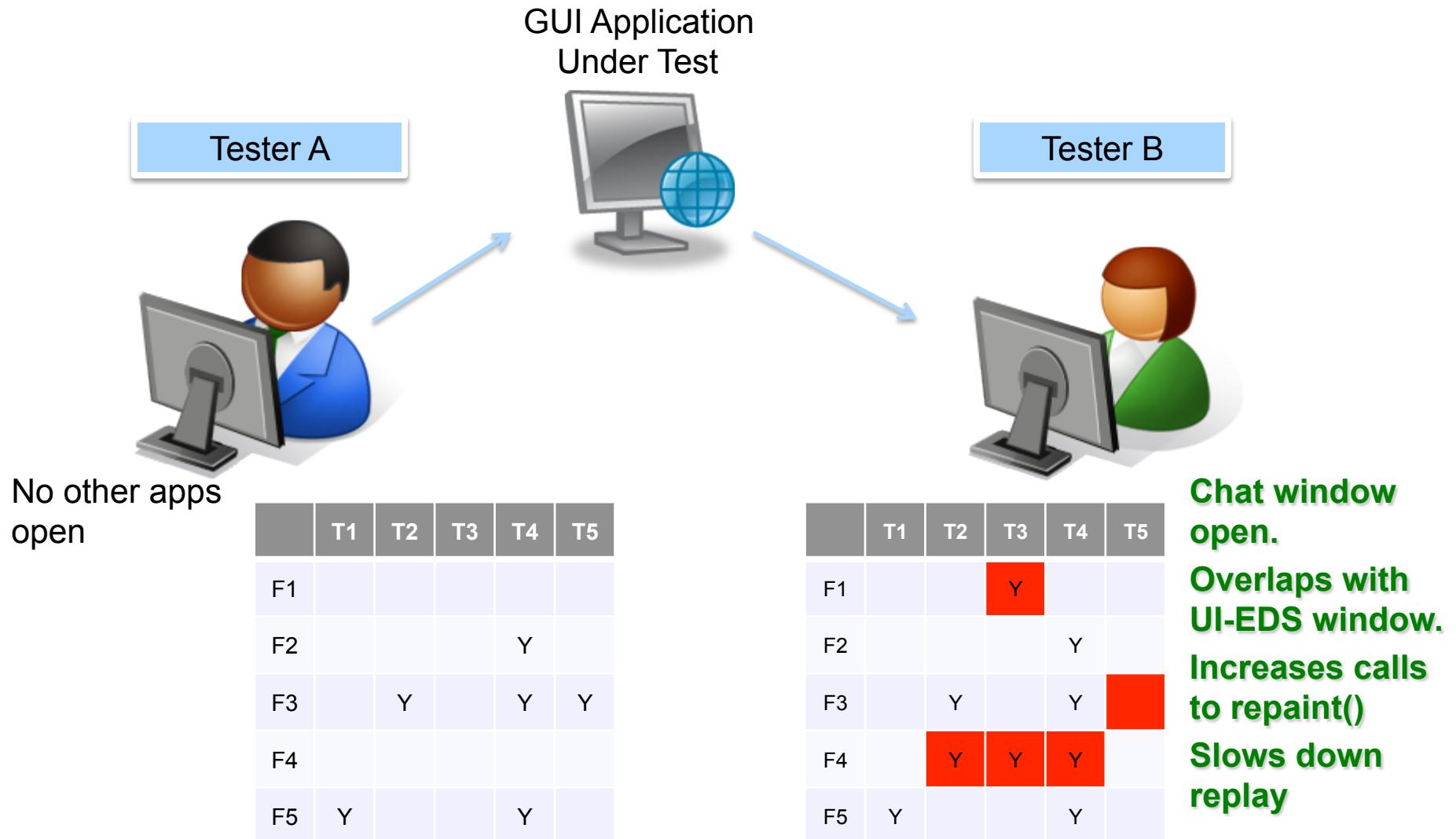
Investigation



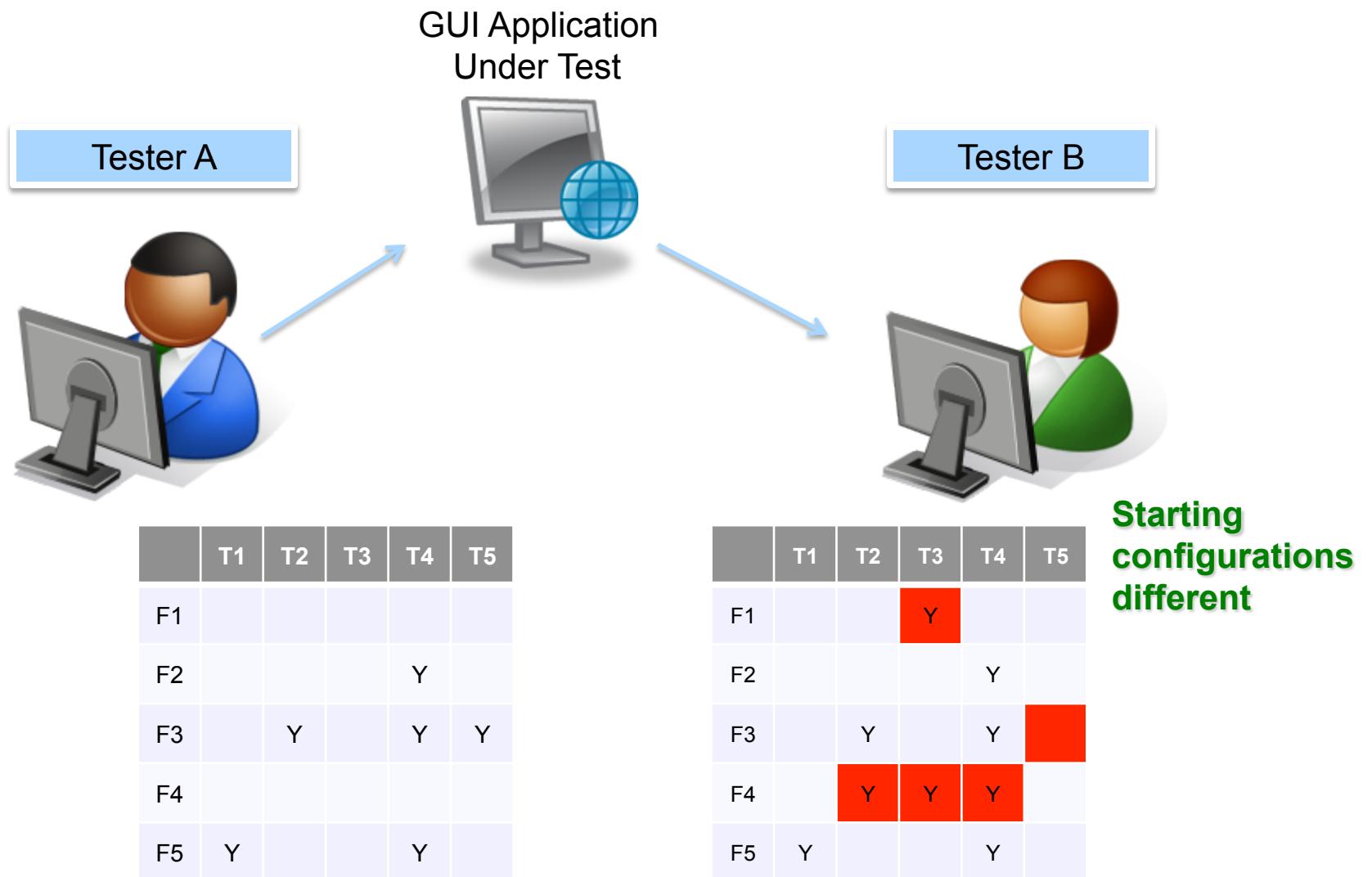
| | T1 | T2 | T3 | T4 | T5 |
|----|----|----|----|----|----|
| F1 | | | | | |
| F2 | | | Y | | |
| F3 | | Y | | Y | Y |
| F4 | | | | | |
| F5 | Y | | | Y | |

| | T1 | T2 | T3 | T4 | T5 |
|----|----|----|----|----|----|
| F1 | | | Y | | |
| F2 | | | | Y | |
| F3 | | Y | | Y | |
| F4 | | Y | Y | Y | |
| F5 | Y | | | Y | |

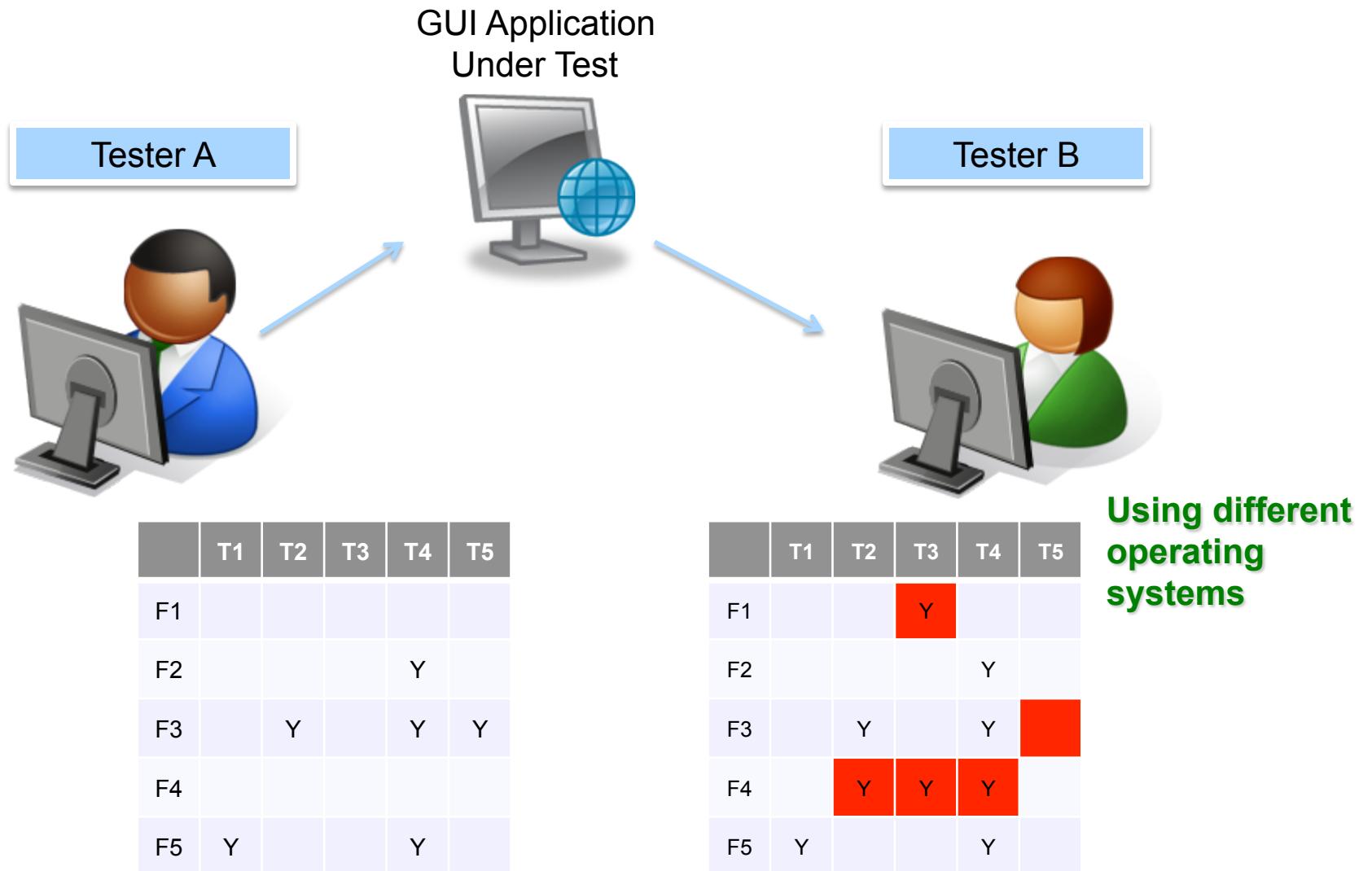
Investigation



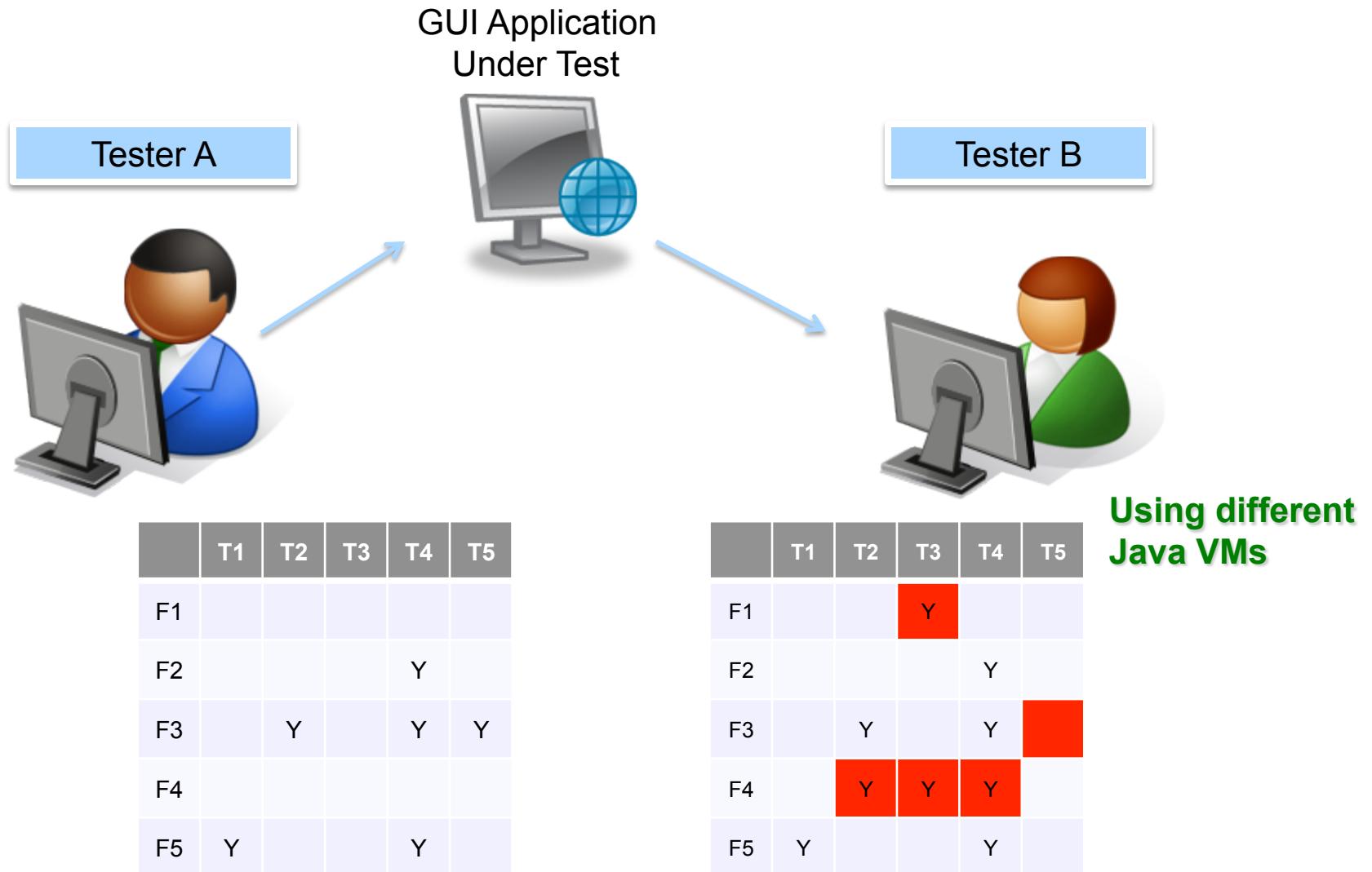
Investigation



Investigation



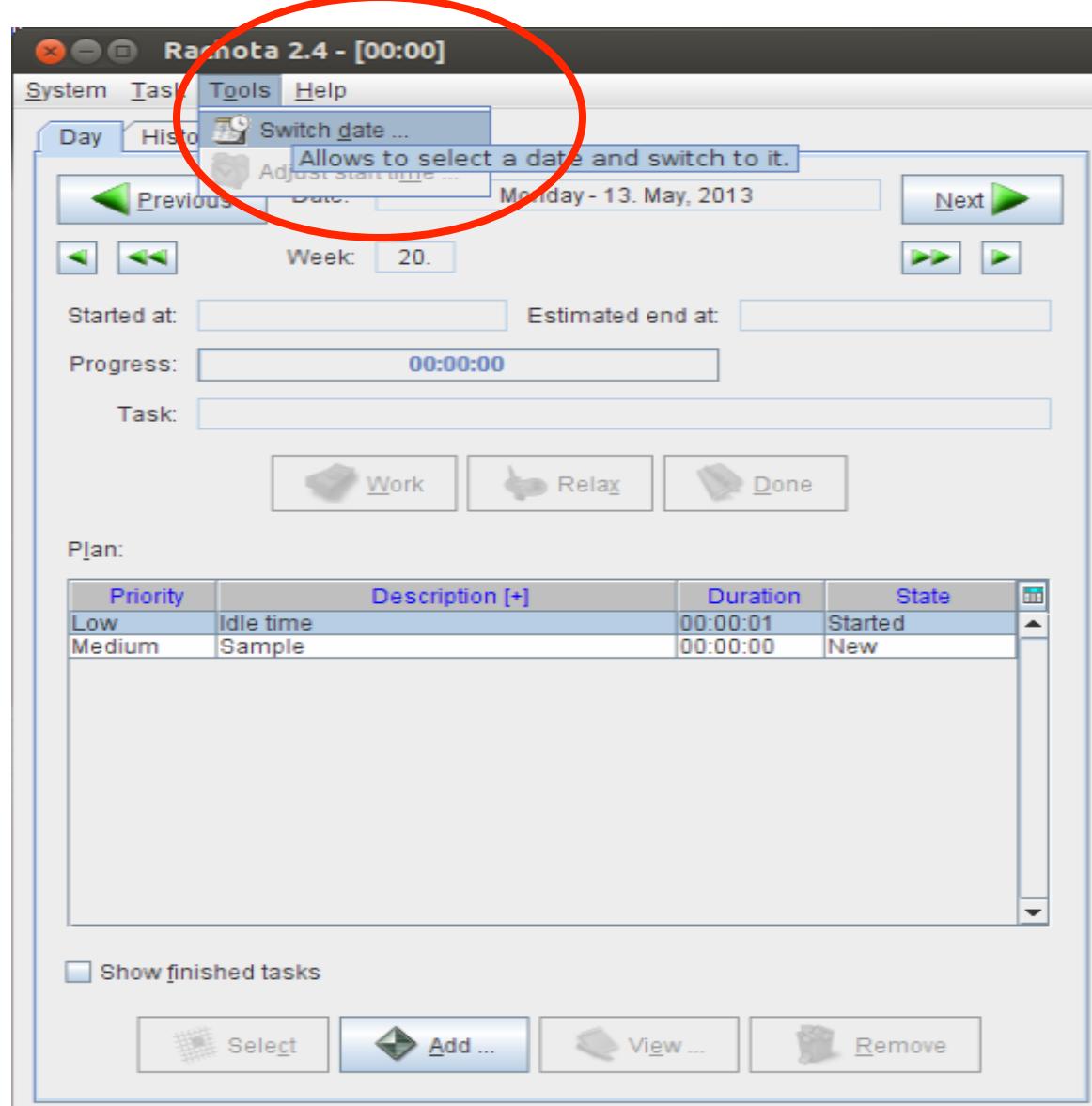
Investigation



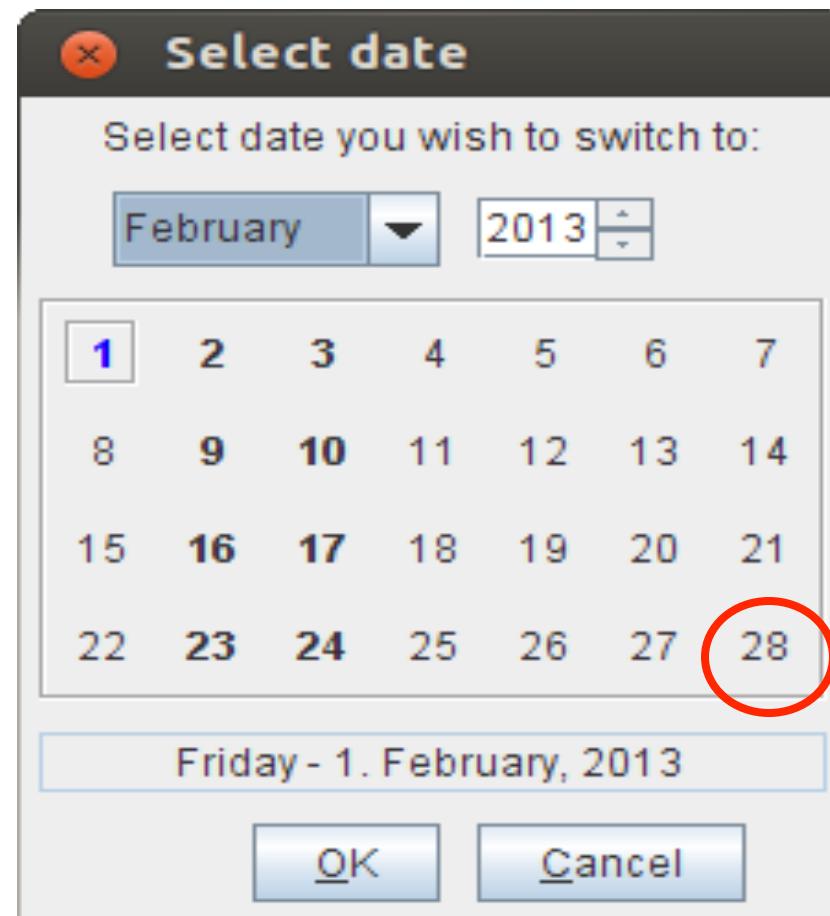
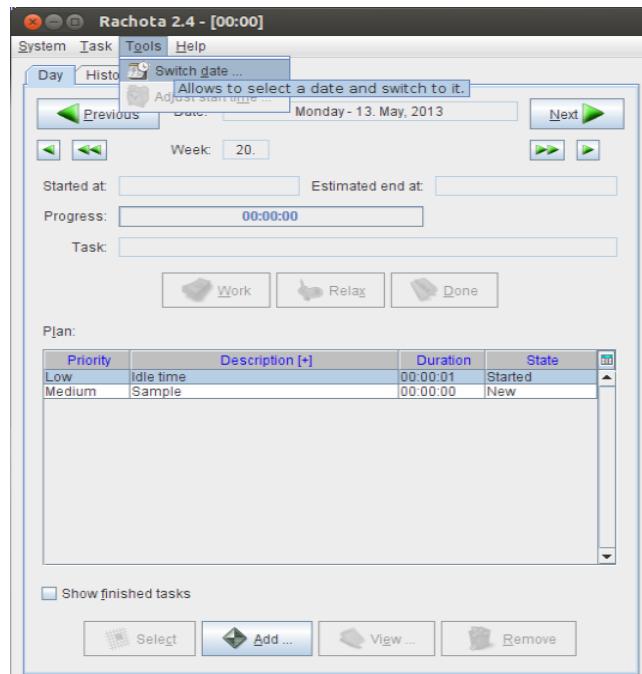
Sources of Variability

- Testing tool (e.g. GUITAR) options
- Application configuration
- External Data
- OS versions, Java
- System/Environment

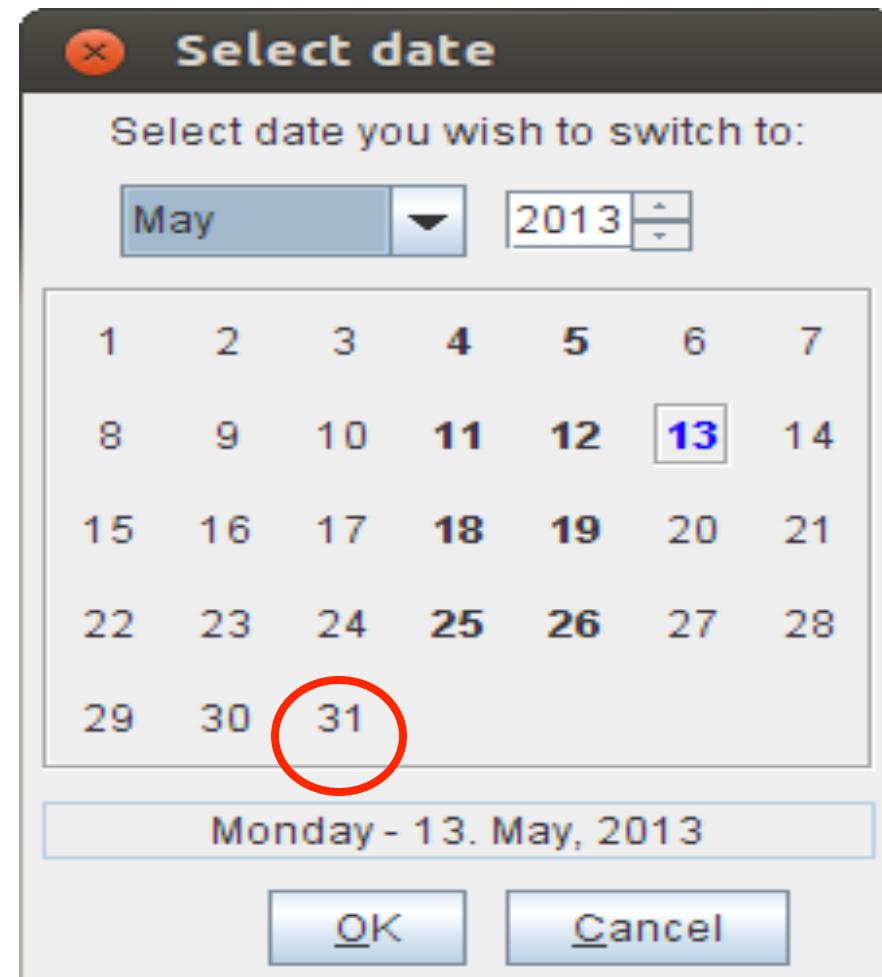
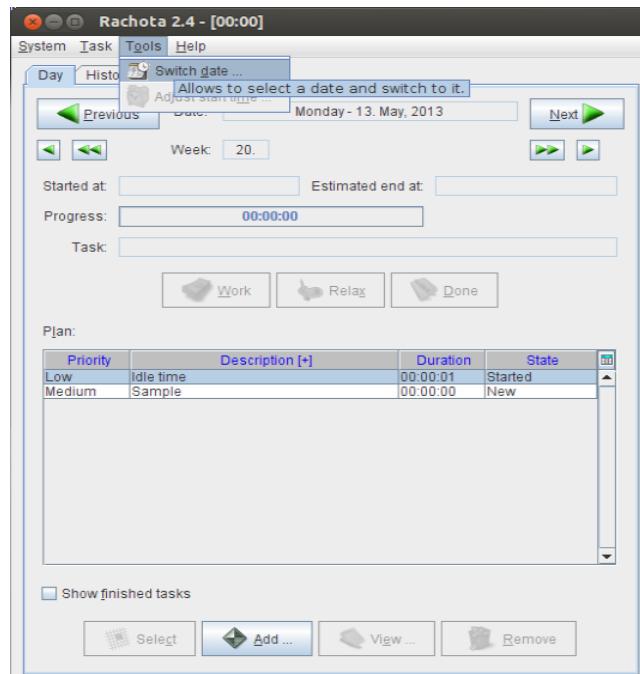
Example: External Input Data



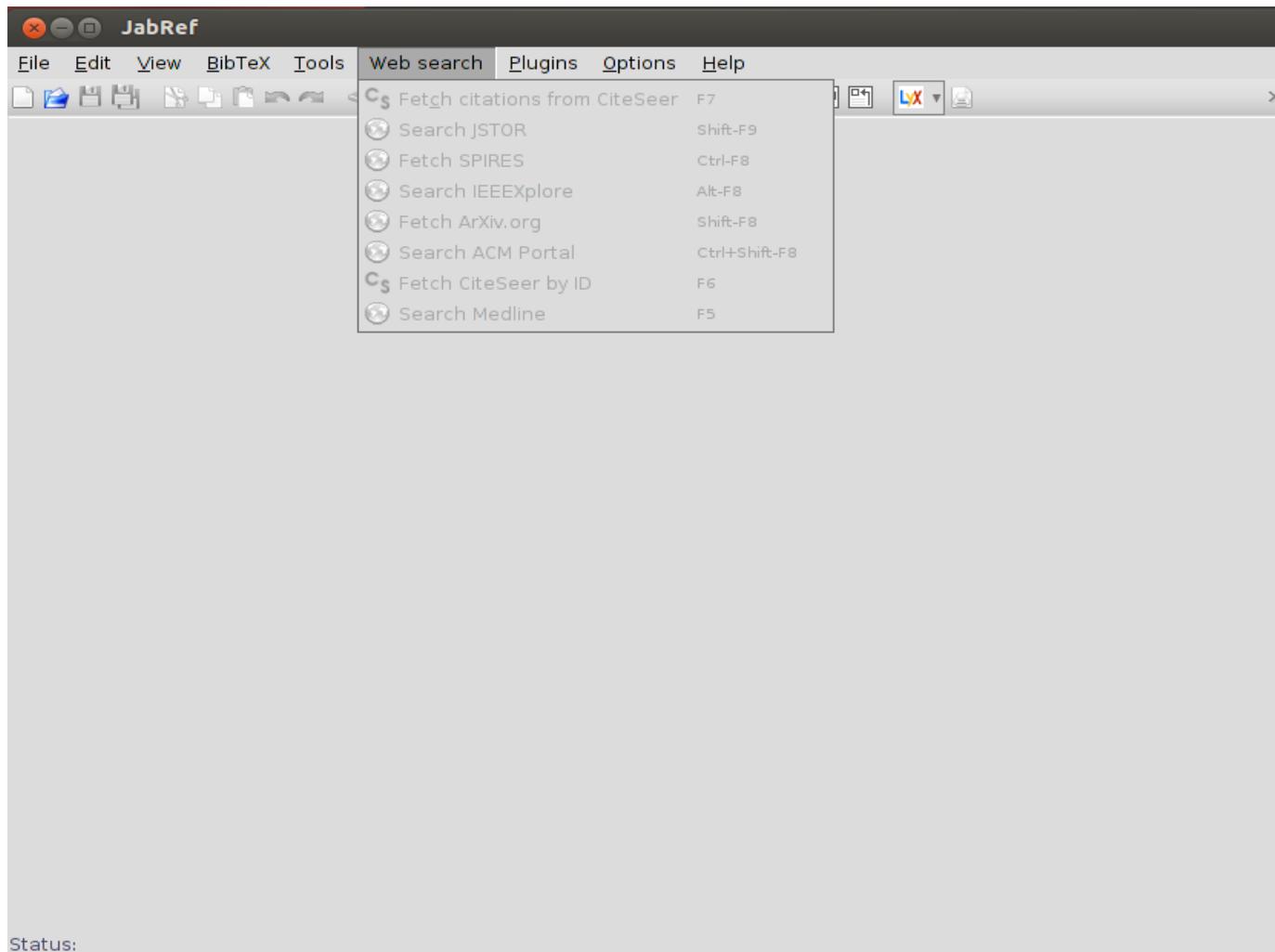
Example: External Input Data



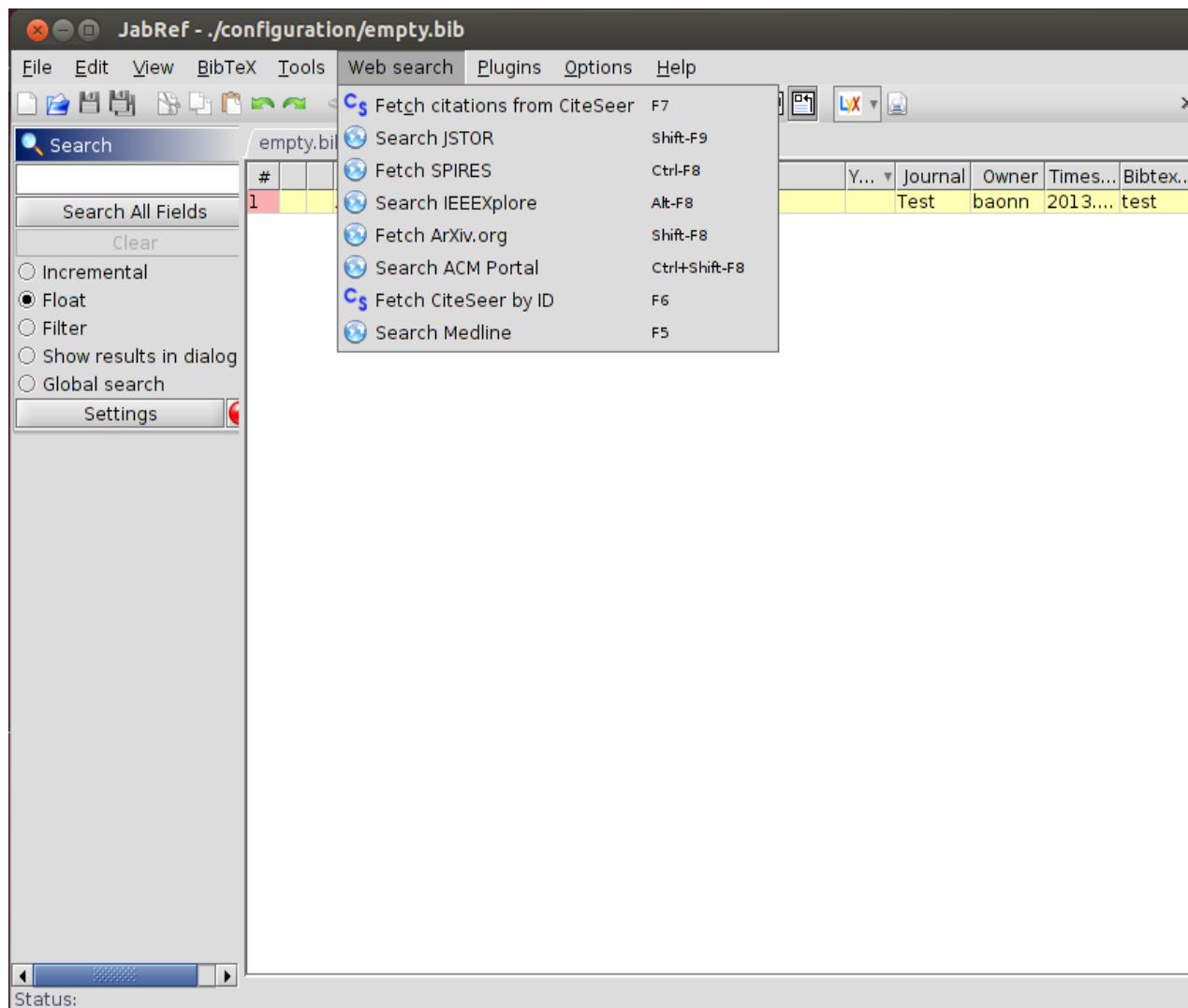
Example: External Input Data



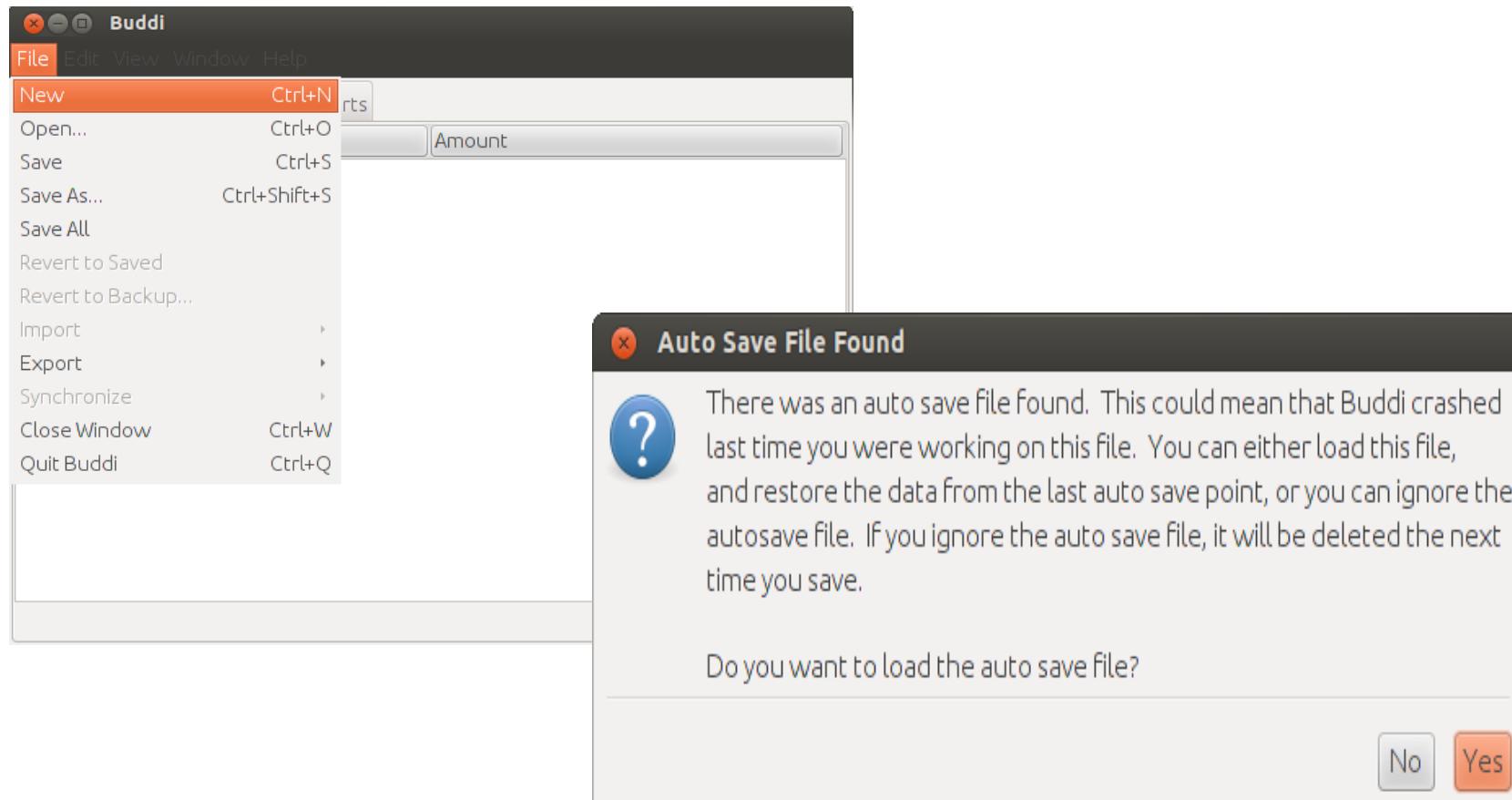
Initial Application State



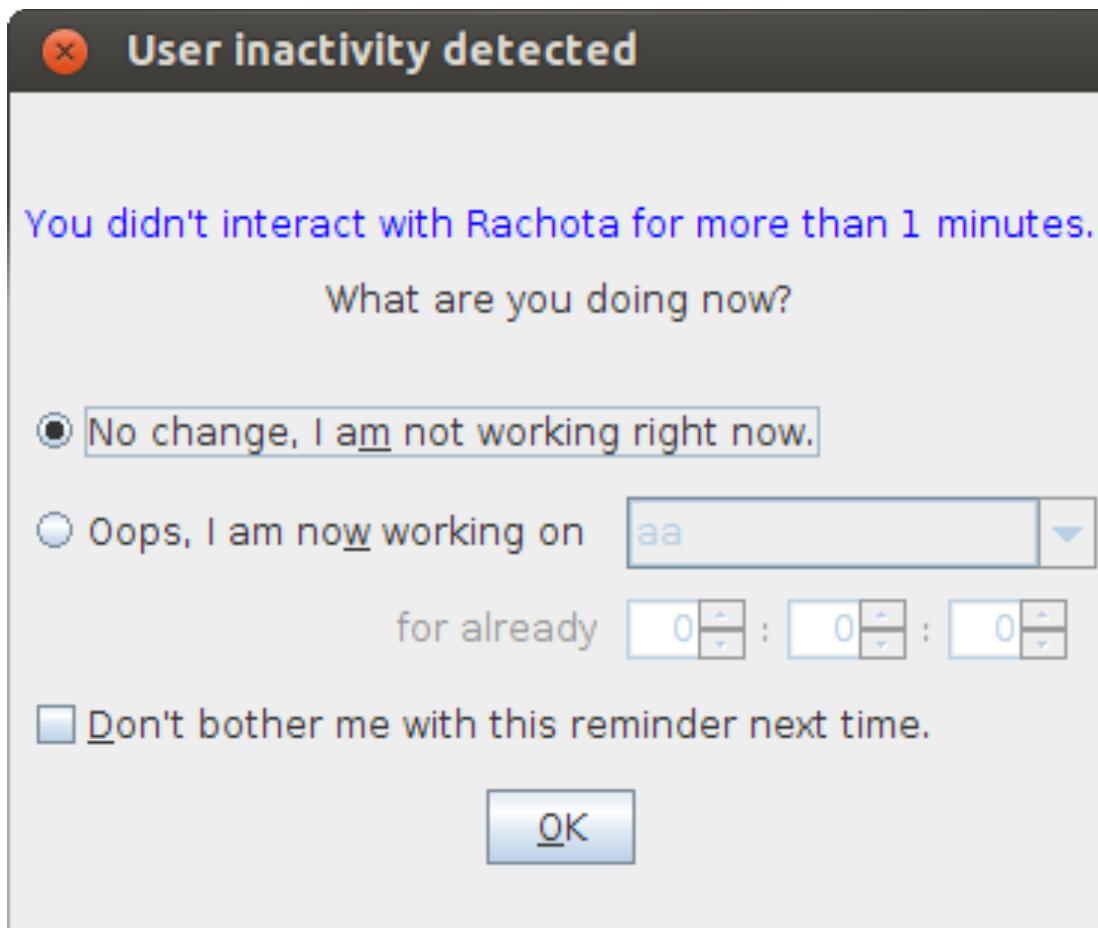
Initial Application State



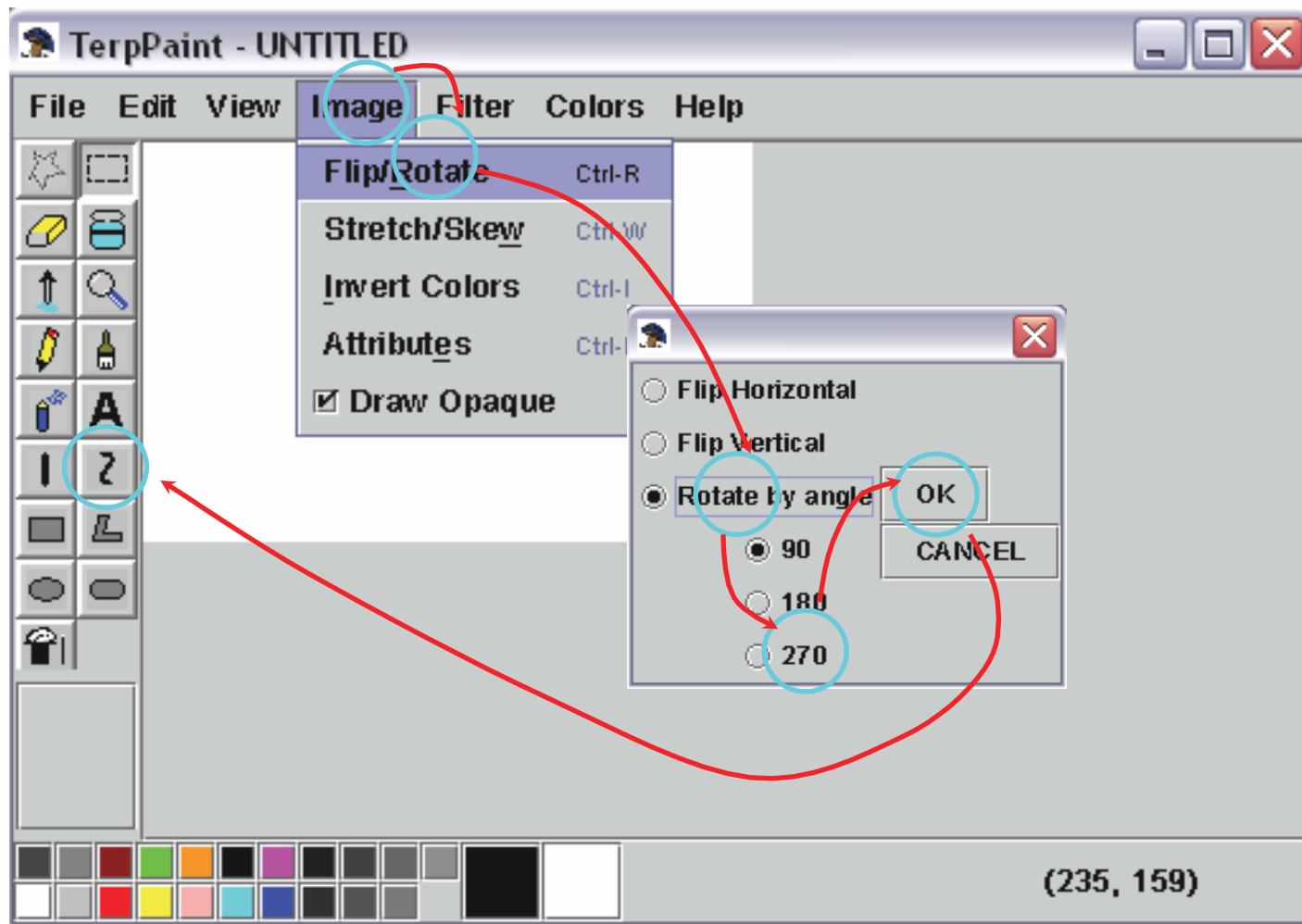
Initial State



Timing Issues



How GUI Test Harnesses Work



Test Harness for GUIs are Complex

- Harness needs to
 - Mimic user's inputs
 - Iterate
 - Read user event from test file
 - » e.g., click button, open menu, close window
 - Execute event
 - Wait for software to finish repainting all widgets
 - Obtain output for verification
 - After each event's execution
 - Extract state of EDS for test oracle
 - » e.g., via screen scraping, windowing API, specialized probes
 - Actual output varies
 - When to collect output?
 - Environment

Test Harness for UI-EDS

- Relies on low-level system calls
 - Event-execution and state-extraction code specialized for specific platform
 - e.g., Java Swing, Java Standard Widget Toolkit, Microsoft Foundation Classes, Win32 API
 - Harnesses developed for each platform on which software tested
- “**Implicit parallelism**”
 - Two separate programs never designed to work together
 - Synchronization and Wait
 - TerpOffice tests broke when Java’s threading policy changed (tool versions -> moving target)

Imagine ...

- Requiring reproducibility in the face of
 - Finicky GUI
 - Complex test harnesses
 - Evolving software tools

Why does this Matter?

- Execution reproducibility is important
 - End users
 - Don't want unexplainable behavior
 - Tester <-> Developer communication
 - Need to reproduce reported faults for debugging
 - Regression testing
 - Previously passed, now failing for no apparent reason
 - “flakey tests”
 - Experimentation
 - Researchers need to compare results

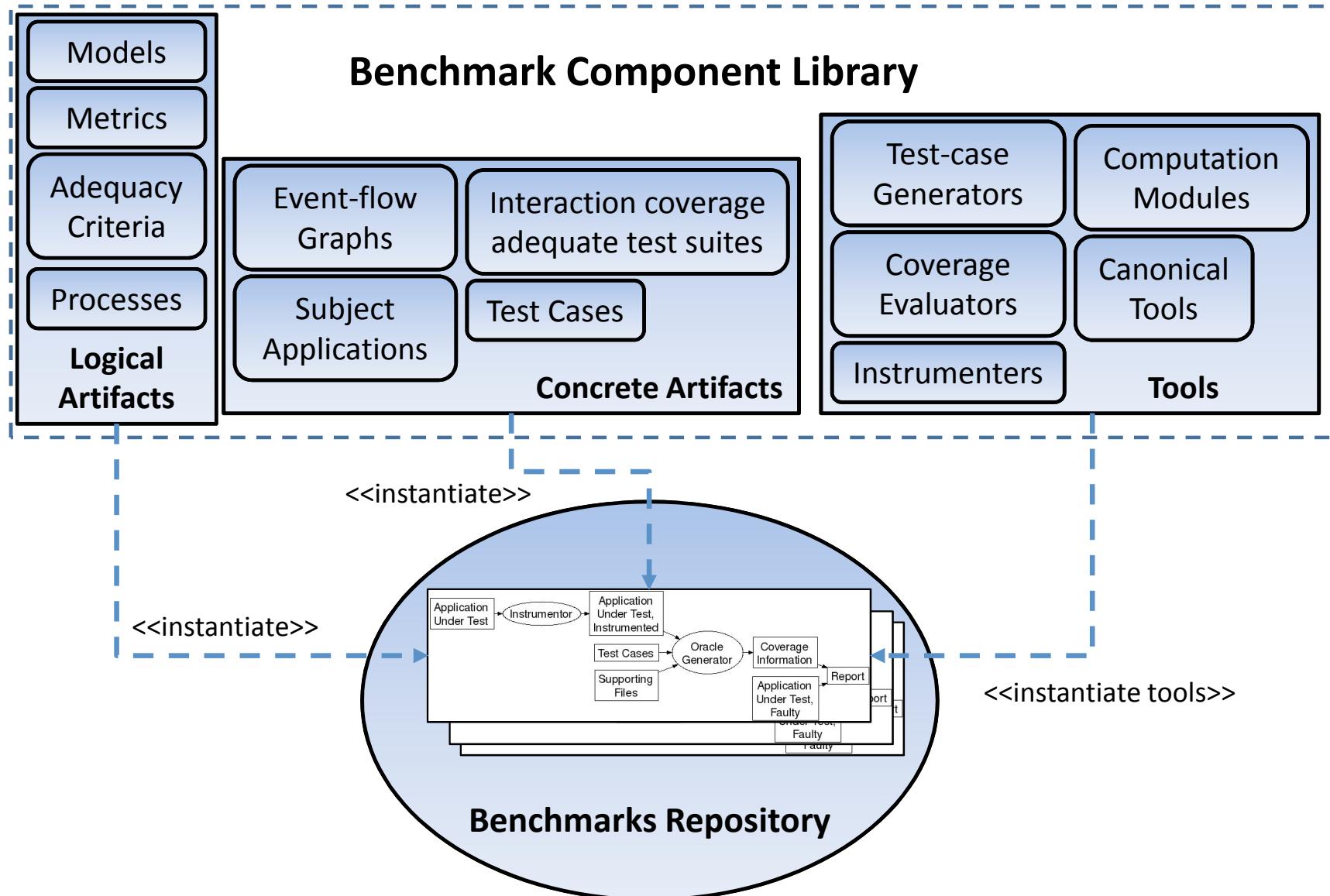
How Can We Reduce Variability?

- Start from the same beginning configurations
- Start tests from clean state
- Use
 - the same OS and VMs
 - consistent system loads
 - the same starting states
 - delays that are long enough (but not too long)
 - the same input data

COMET

- **COMmunity Event-based Testing (COMET)**
 - Broader software testing community
 - <http://comet.unl.edu>
- US National Science Foundation grants
 - CNS-0855139, CNS-0855055, CNS-1205472, CNS-1205501
- Provide benchmarks for others to use
- Provide infrastructure of benchmarks, tools, models, test artifacts
- Our attempt to start to address some of this variability in the research and practitioner community
- Community involved.
- We plan to:
 - Develop Flexible Design of Artifacts
 - Create Methods to Control Variation
 - Design for Evolution

Comet Vision



The Comet Website

The screenshot shows a web browser window displaying the Comet website at comet.unl.edu. The page features a navigation bar with links for Home, About, Benchmarks, Publications, People, Feedback, and Getting Involved. A "Welcome" section discusses event-driven software testing and its applications. A "News" section lists ten milestones from 2009 to 2012. An "Acknowledgments" section credits National Science Foundation grants. Logos for the University of Maryland, NSF, and ESQuaReD are present, along with a guitar icon. The footer includes contact information and a timestamp.

Subjects
Processes
Tools
Models

COMET - Community Event-based Testing

[Home](#) [About](#) [Benchmarks](#) [Publications](#) [People](#) [Feedback](#) [Getting Involved](#) [Register](#) | [Sign in](#)

Welcome

Event-driven software (EDS) spans multiple domains, from industrial embedded devices and robotic controllers to web interfaces and GUI applications. This website will serve as a community infrastructure for event-based testing researchers to provide uniformity in experimentation and benchmarking in event-driven software testing. We will provide concrete artifacts for testing that includes subjects, test suites, fault matrices and tools, as well as processes and models to standardize the way experiments are conducted in this environment.

This project is a joint effort between the [E2 laboratory at UNL](#) and the [GUITAR group at UMD](#).

News

10/22/2012 We have received funding to keep COMET going. Look for new updates and many more benchmarks in the coming months!

05/08/2012 Comet tools were used in a paper presented at [CHI 2012](#) this week. [Paper](#).

03/29/2011 Our first community contributed benchmark has been released. [SAPE-Pounder-2010](#).

03/15/2011 [TESTBEDS '11](#): The third international workshop on TESTing Techniques & Experimentation Benchmarks for Event-Driven Software to be held in Berlin in March in association with [ICST '11](#)

06/12/2010 Check out our new benchmark collection [UMD.Reduction.TSE.2008](#).

04/01/2010 Our first two benchmark collections have been released!

11/05/2009 [TESTBEDS '10](#): The second international workshop on TESTing Techniques & Experimentation Benchmarks for Event-Driven Software to be held in Paris in April in association with [ICST '10](#).

11/04/2009 Preliminary version of the web site released.

10/30/2009 This project gets its domain at <http://comet.unl.edu>.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. [CNS-0855139](#) and [CNS-0855055](#), [CNS-1205472](#) and [CNS-1205501](#).

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Contact us: comet@cse.unl.edu
Last modified: Oct. 22, 2012, 13:09:59, CDT

Benchmark Collections

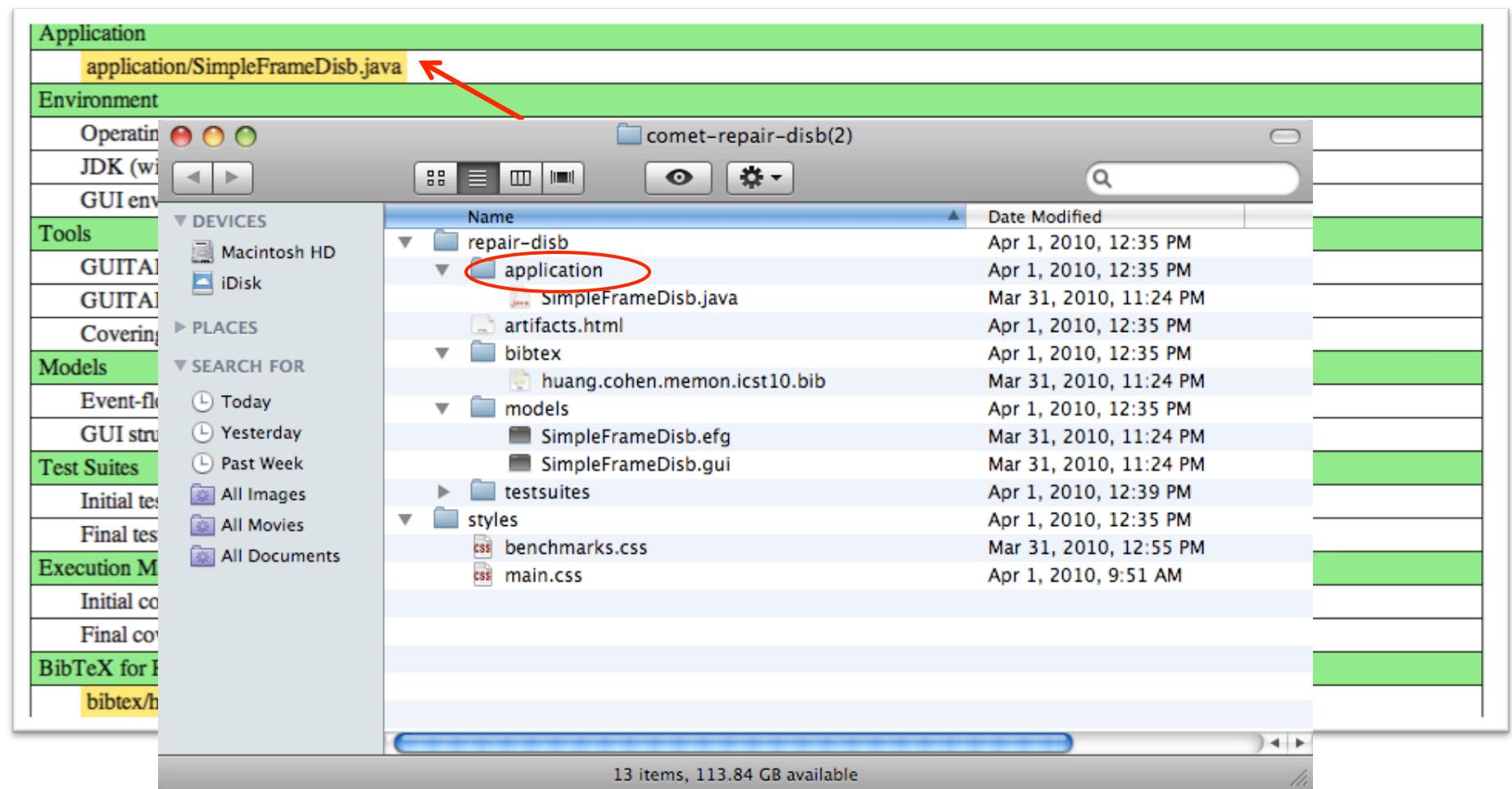
Benchmarks

Our benchmarks are grouped as collections. The collections represent logical groupings from a specific contributor. Within collection there may be several individual programs. You will have the opportunity to select all or part of each benchmark for download. When you expand the benchmark you will be able to view information for each benchmark that includes its operating system environment, html links to tools required to run/use the benchmark, and descriptions of which test suites and execution matrices are included. The highlighted golden text indicates a relative path in the benchmark folder. When the benchmark is the result of a publication, the bib file for that publication will be included as well.

Views: [Collections](#) [Benchmarks](#)

| | Collection Name | Description |
|--------------------------|--|--|
| <input type="checkbox"/> | UNL.Toy.2010 | <p>This collection of benchmarks was used in the experiments for GUI test suite repair in the paper <i>Repairing GUI Test Suites Using a Genetic Algorithm</i>, published in ICST 2010. The website with full results for this paper is http://cse.unl.edu/~myra/artifacts/icst2010. A bibtex entry for this paper is attached with each benchmark, or you can use the following.</p> <pre>@inproceedings{huang.cohen.memon.icst10, author={Si Huang and Myra B. Cohen and Atif M. Memon}, title={Repairing {GUI} Test Suites Using a Genetic Algorithm}, booktitle={ICST '10: International Conference on Software Testing, Verification and Validation}, address={Paris, France}, month={April}, year={2010} }</pre> |
| <input type="checkbox"/> | UMD.Reduction.TSE.2008 | <p>This collection of benchmarks was used in the paper <i>McMaster, S. and Memon, A. M. Call stack coverage for GUI test-suite reduction. IEEE Trans. Softw. Eng., 2008, IEEE Press.</i>. The users can use those benchmarks to evaluate different testsuite reduction techniques.</p> <pre>@article{McMasterMemonTSE2008, author = {Scott McMaster and Atif M. Memon}, title = {Call-Stack Coverage for {GUI} Test-Suite Reduction }, journal = {IEEE Trans. Softw. Eng.}, publisher = {IEEE Press}, address = {Piscataway, NJ, USA}, year = {2008} }</pre> |

Benchmark Structure



Benchmark Structure

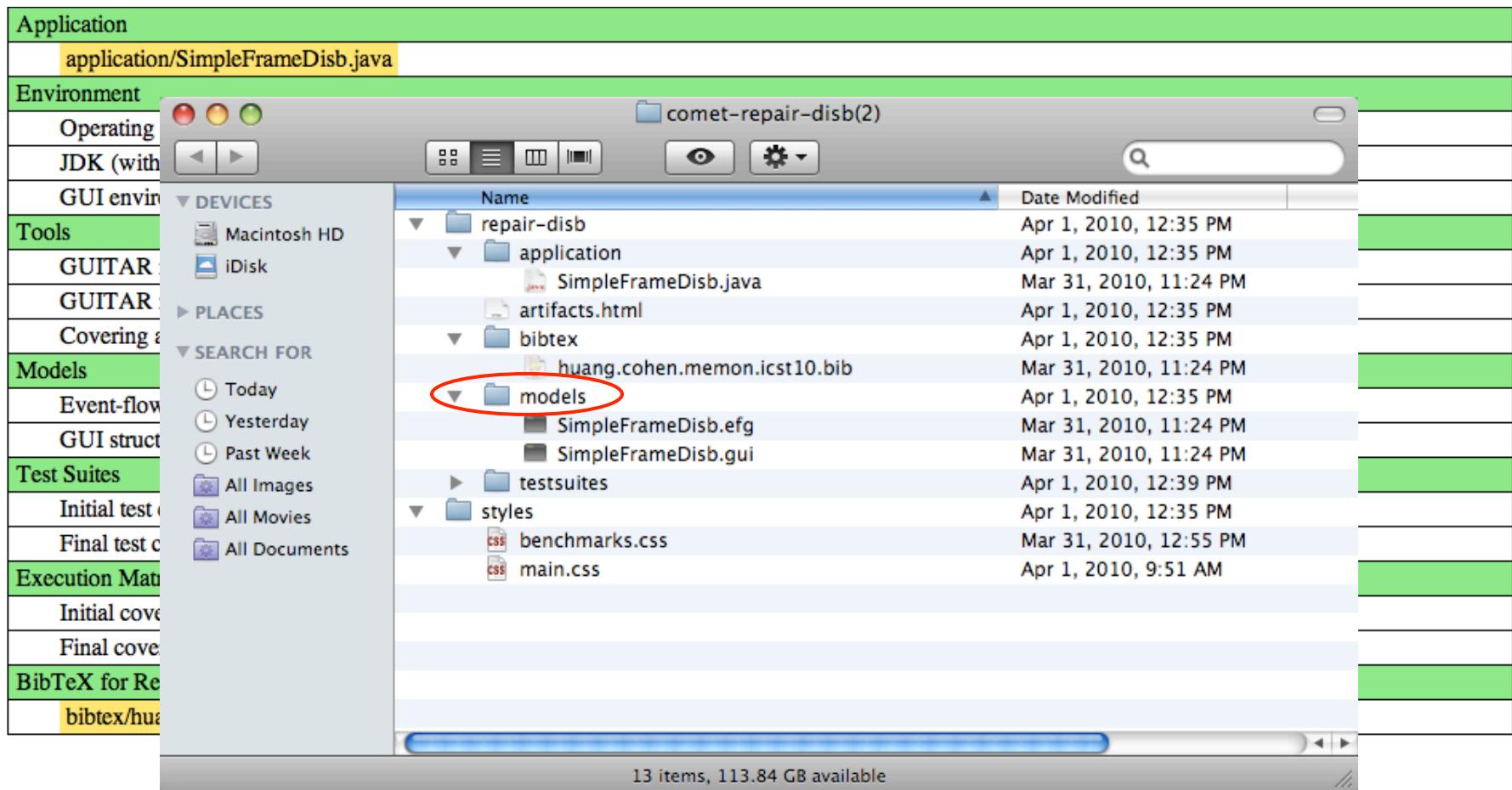
| | |
|-------------------------------------|---|
| Application | application/SimpleFrameDisb.java |
| Environment | |
| Operating system: | Linux 2.6.18 (Fedora Core) |
| JDK (with JRE): | Java 1.6 update 16 |
| GUI environment: | X virtual framebuffer (Xvfb) |
| Tools | |
| GUITAR ripper: | http://www.cs.umd.edu/~atif/Benchmarks/common/JavaGUIReplayer2.zip |
| GUITAR replayer: | http://guitar.svn.sourceforge.net/viewvc/guitar/GUITestRunner/tags/v1.1/ |
| Covering array generator: | http://www.cse.unl.edu/citportal/tools/casa/ |
| Models | |
| Event-flow graph (EFG): | models/SimpleFrameDisb.efg |
| GUI structure: | models/SimpleFrameDisb.gui |
| Test Suites | |
| Initial test cases (before repair): | testsuites/initial.zip |
| Final test cases (after repair): | testsuites/final.zip |
| Execution Matrices | |
| Initial coverage (before repair): | testsuites/initialexemat.zip |
| Final coverage (after repair): | testsuites/finalexemat.zip |
| BibTeX for Reference | |
| bibtex/huang.cohen.memon.icst10.bib | |

Benchmark Structure

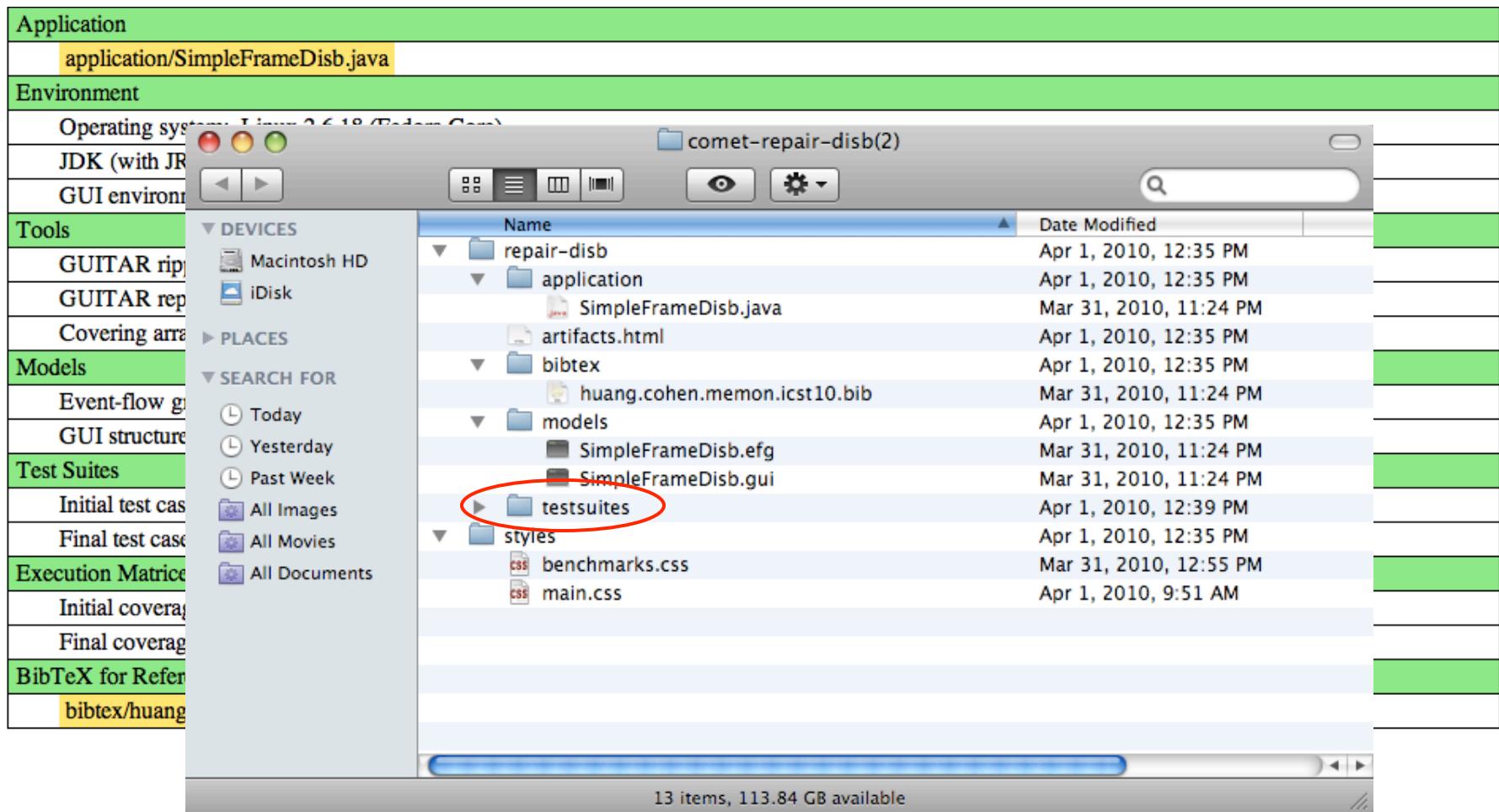
| |
|--|
| Application |
| application/SimpleFrameDisb.java |
| Environment |
| Operating system: Linux 2.6.18 (Fedora Core) |
| JDK (with JRE): Java 1.6 update 16 |
| GUI environment: X virtual framebuffer (Xvfb) |
| Tools |
| GUITAR ripper: http://www.cs.umd.edu/~atif/Benchmarks/common/JavaGUIReplayer2.zip |
| GUITAR replayer: http://guitar.svn.sourceforge.net/viewvc/guitar/GUITestRunner/tags/v1.1/ |
| Covering array generator: http://www.cse.unl.edu/citportal/tools/casa/ |
| Models |
| Event-flow graph (EFG): models/SimpleFrameDisb.efg |
| GUI structure: models/SimpleFrameDisb.gui |
| Test Suites |
| Initial test cases (before repair): testsuites/initial.zip |
| Final test cases (after repair): testsuites/final.zip |
| Execution Matrices |
| Initial coverage (before repair): testsuites/initialexemat.zip |
| Final coverage (after repair): testsuites/finalexemat.zip |
| BibTeX for Reference |
| bibtex/huang.cohen.memon.icst10.bib |



Benchmark Structure

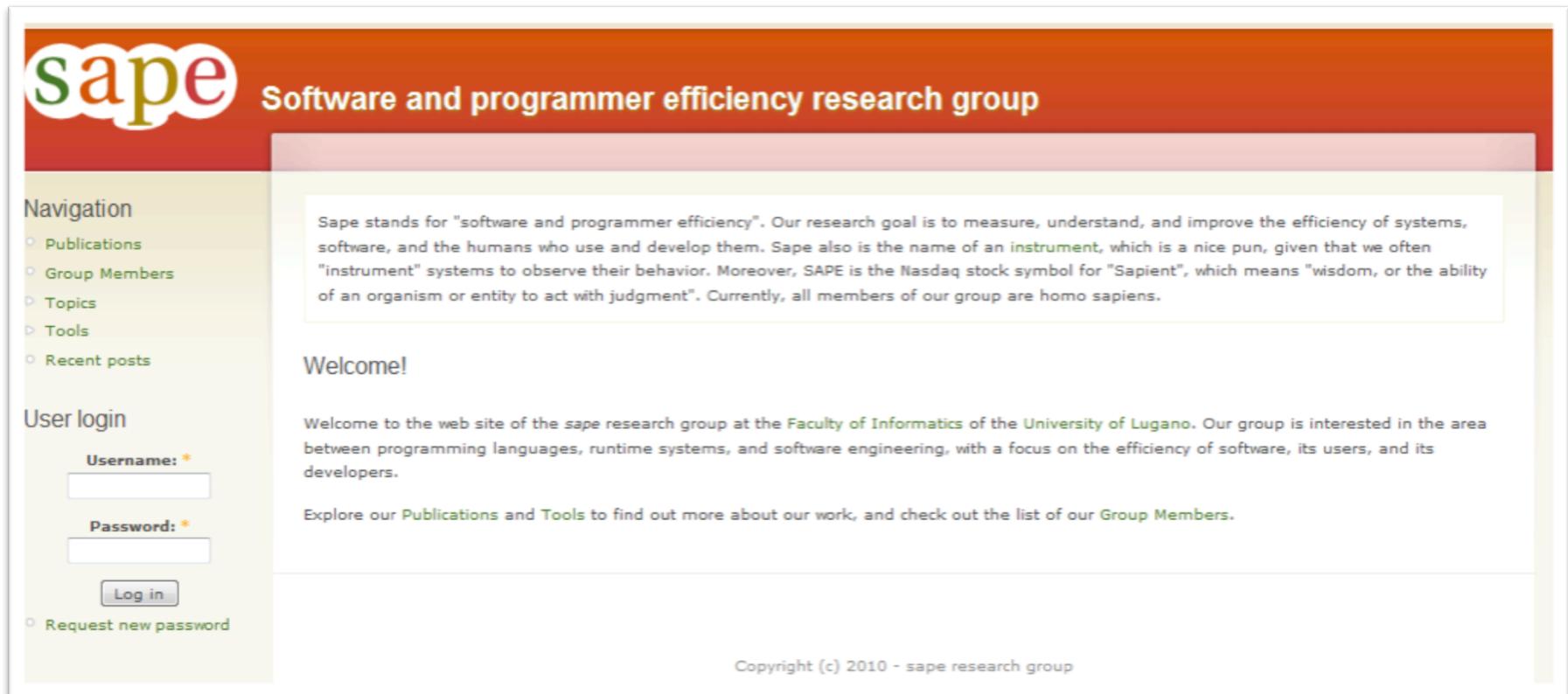


Benchmark Structure



Community Involvement

- Software and Programmer Efficiency Research Group (Hauswirth,M; Adamoli, A; Zaparanuks, D; Jovic, M)
AST'10: Automating Performance Testing of Interactive Java Applications



The screenshot shows the homepage of the Sape research group. The header features a large, stylized logo "sape" in white, with each letter having a different color: S is blue, a is red, p is green, and e is yellow. To the right of the logo, the text "Software and programmer efficiency research group" is displayed in a white, sans-serif font against a red-to-orange gradient background.

Navigation

- Publications
- Group Members
- Topics
- Tools
- Recent posts

User login

Username: *

Password: *

Request new password

Sape stands for "software and programmer efficiency". Our research goal is to measure, understand, and improve the efficiency of systems, software, and the humans who use and develop them. Sape also is the name of an [instrument](#), which is a nice pun, given that we often "instrument" systems to observe their behavior. Moreover, SAPE is the Nasdaq stock symbol for "Sapient", which means "wisdom, or the ability of an organism or entity to act with judgment". Currently, all members of our group are homo sapiens.

Welcome!

Welcome to the web site of the [sape](#) research group at the [Faculty of Informatics](#) of the [University of Lugano](#). Our group is interested in the area between programming languages, runtime systems, and software engineering, with a focus on the efficiency of software, its users, and its developers.

Explore our [Publications](#) and [Tools](#) to find out more about our work, and check out the list of our [Group Members](#).

Copyright (c) 2010 - sape research group

Summary

- Getting consistent testing results in GUIs is not-trivial
- Need to control for as many factors as we can to ensure repeatability
- Best way to do this is still an open question
- We are working on COMET – need more community input/feedback so that we can make it useful

This material is based upon work supported by the National Science Foundation under Grant No. [CNS-0855139](#) and [CNS-0855055](#). [CNS-1205472](#) and [CNS-1205501](#). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

ICSE 2013 Tutorial

Automated Testing of GUI Applications Models, Tools and Controlling Flakiness



Atif M. Memon and Myra B. Cohen

UNIVERSITY OF
Nebraska
Lincoln