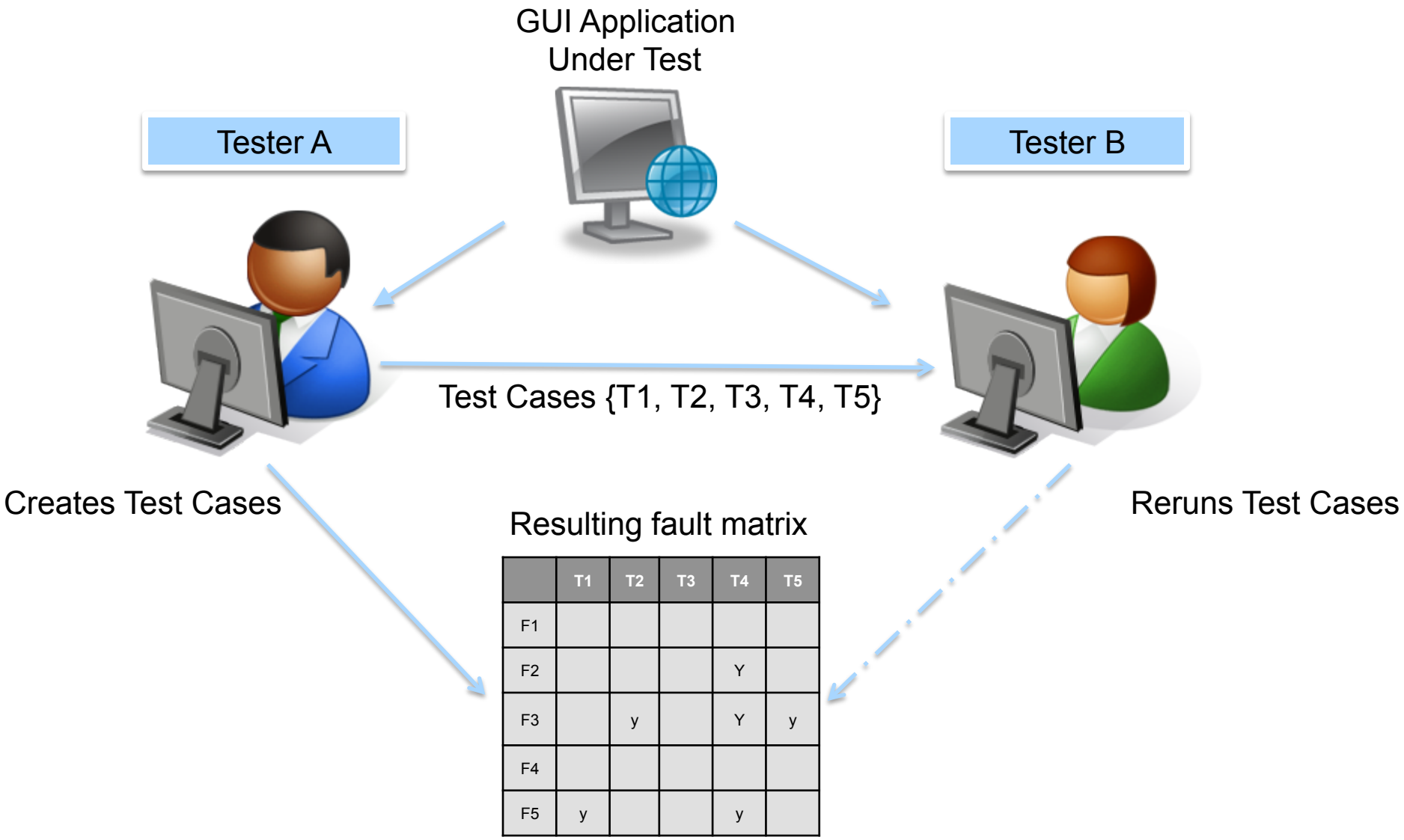
Controlling Factors

ICSE 2013 Tutorial

Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Atif M. Memon and Myra B. Cohen

A Tale of Two Testers



	T1	T2	T3	T4	T5
F1					
F2				Y	
F3		y		Y	y
F4					
F5	y			y	

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

GUI Application Under Test

Tester A

Tester B

No other apps open

	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

GUI Application Under Test

Tester A

Tester B

No other apps open

	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	

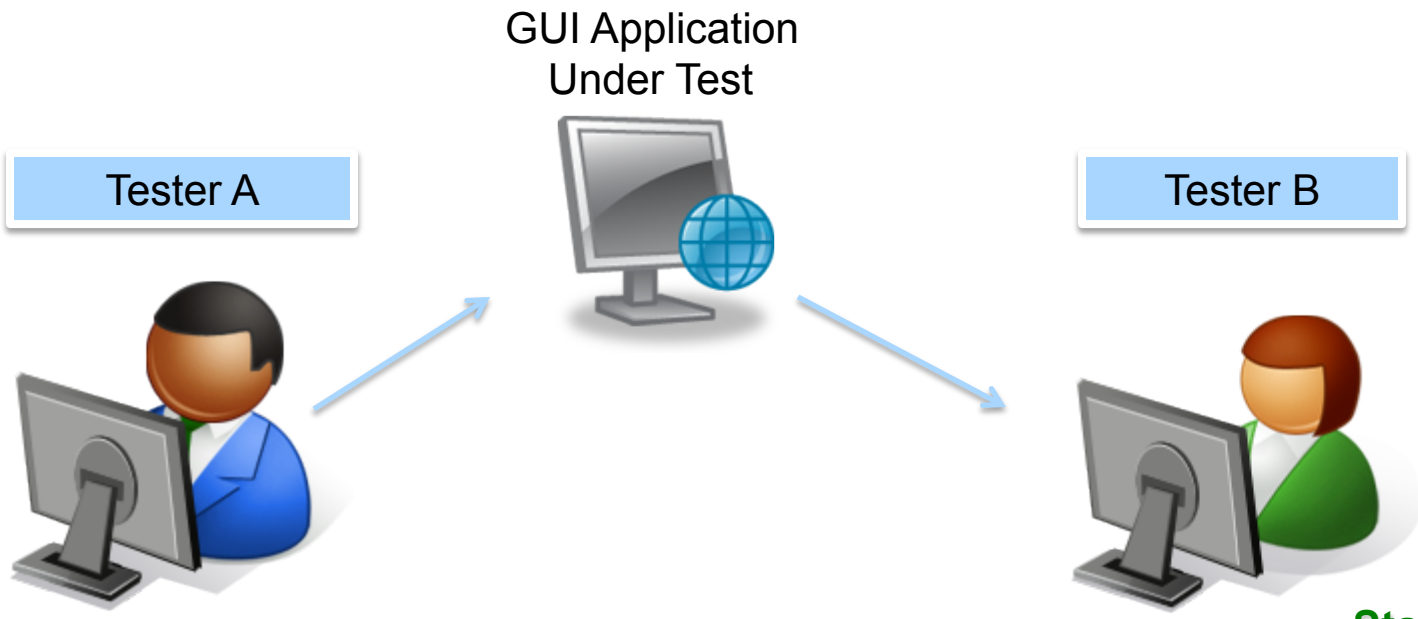
Chat window open.

Overlaps with UI-EDS window.

Increases calls to repaint()

Slows down replay

Investigation

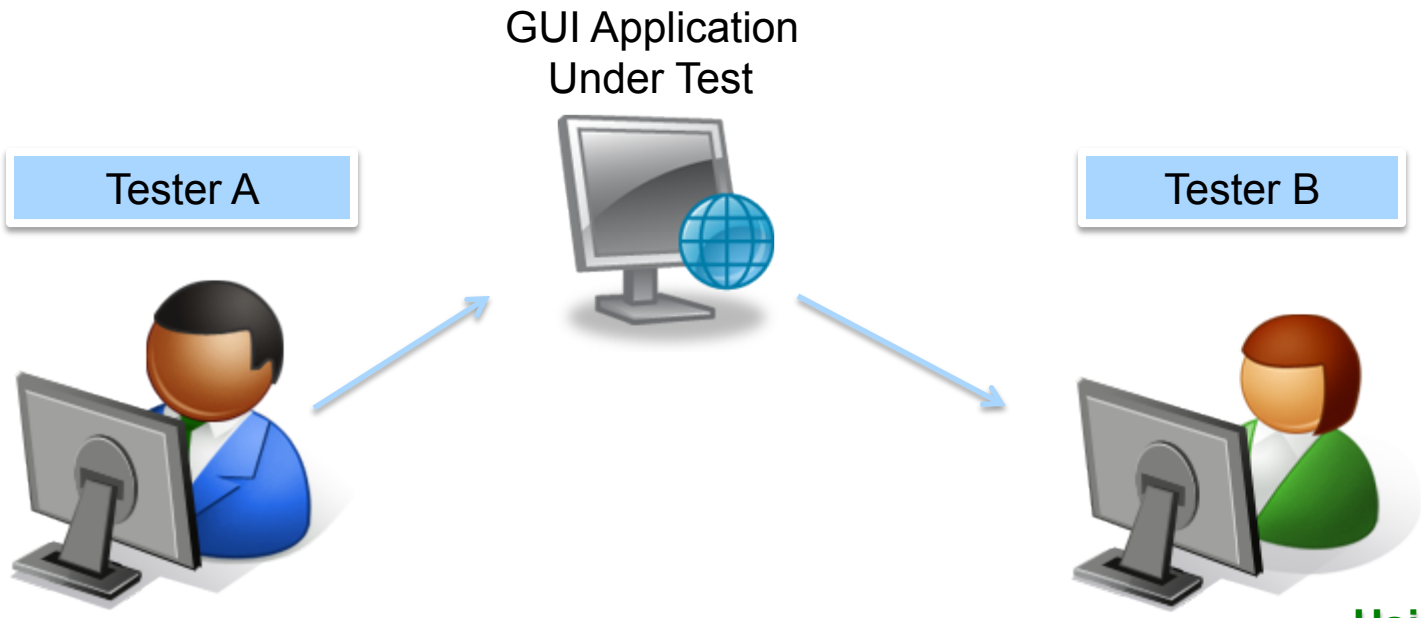


	T1	T2	T3	T4	T5
F1					
F2				Y	
F3		Y		Y	Y
F4					
F5	Y			Y	

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

Starting configurations different

Investigation

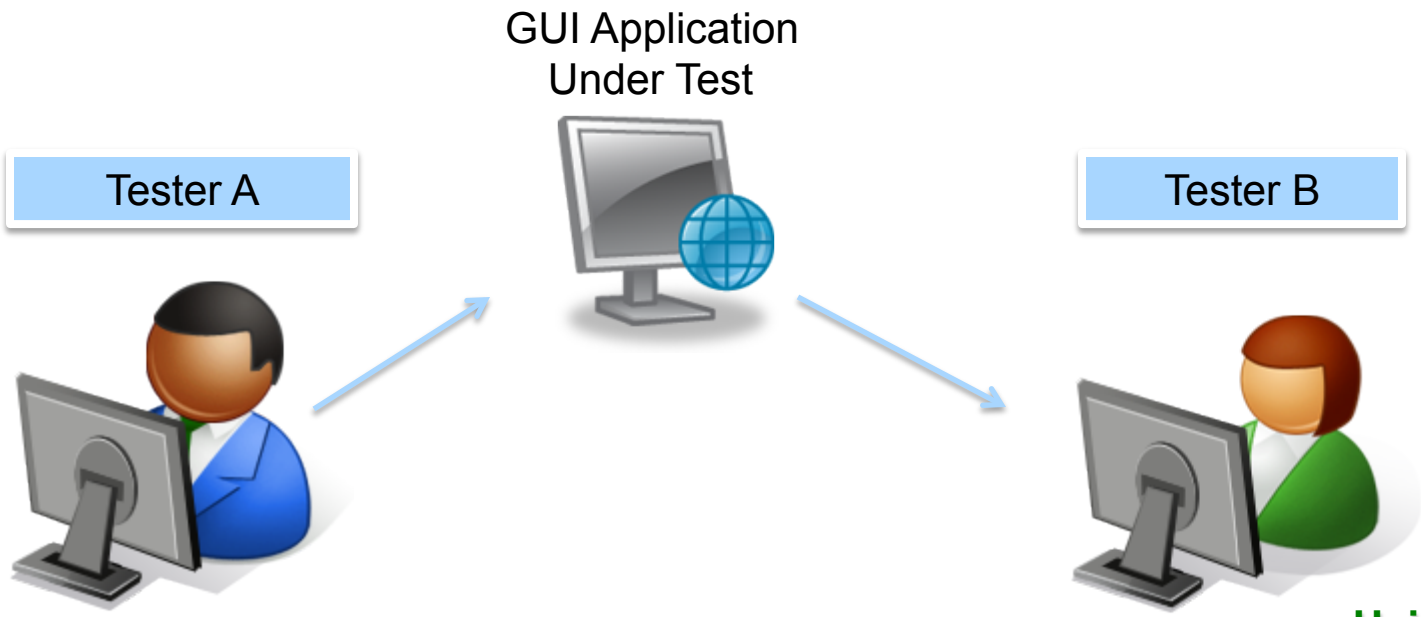


	T1	T2	T3	T4	T5
F1					
F2				Y	
F3		Y		Y	Y
F4					
F5	Y			Y	

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

Using different operating systems

Investigation



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

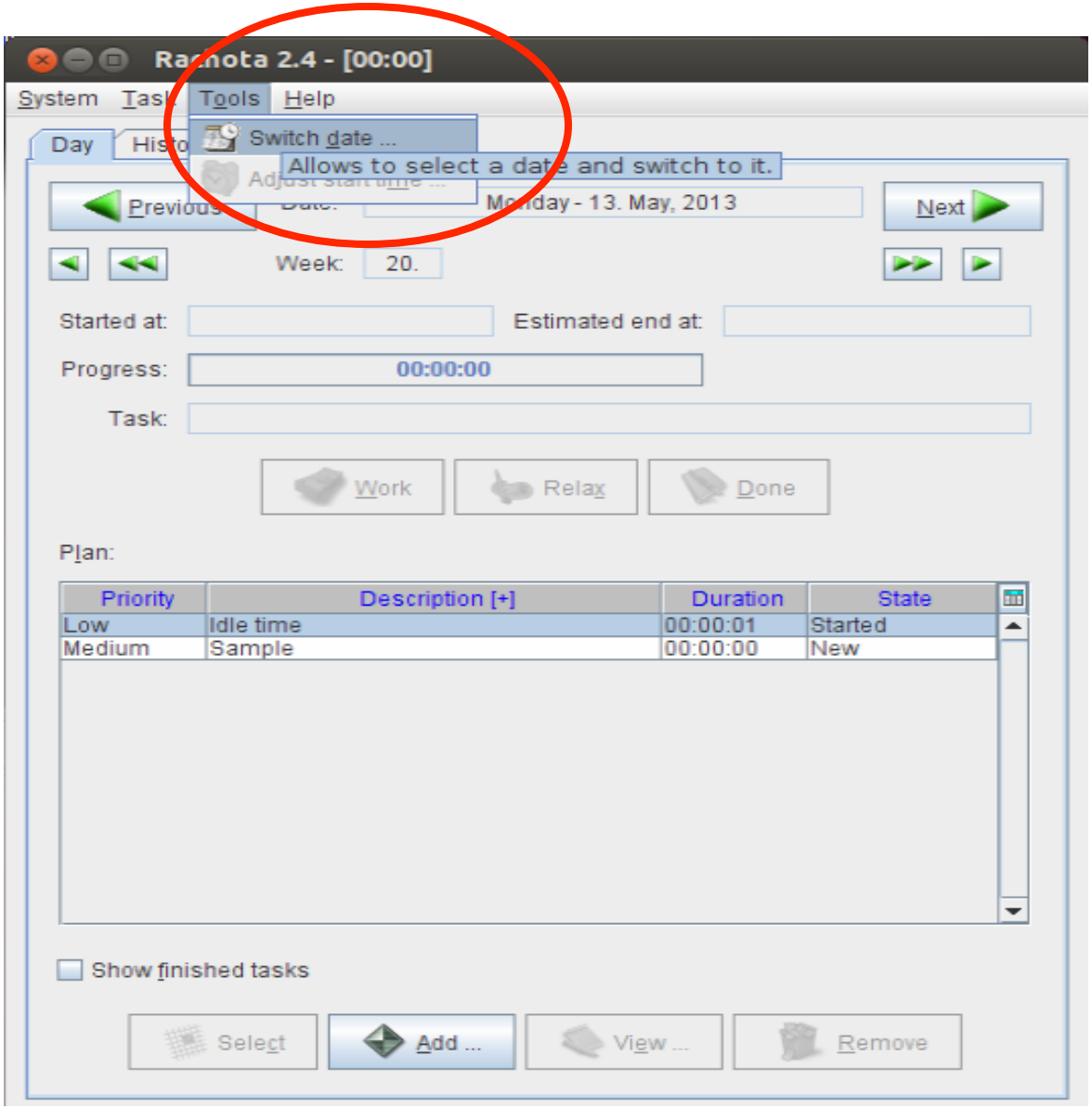
	T1	T2	T3	T4	T5
F1			Y		
F2				Y	
F3		Y		Y	
F4		Y	Y	Y	
F5	Y			Y	

Using different Java VMs

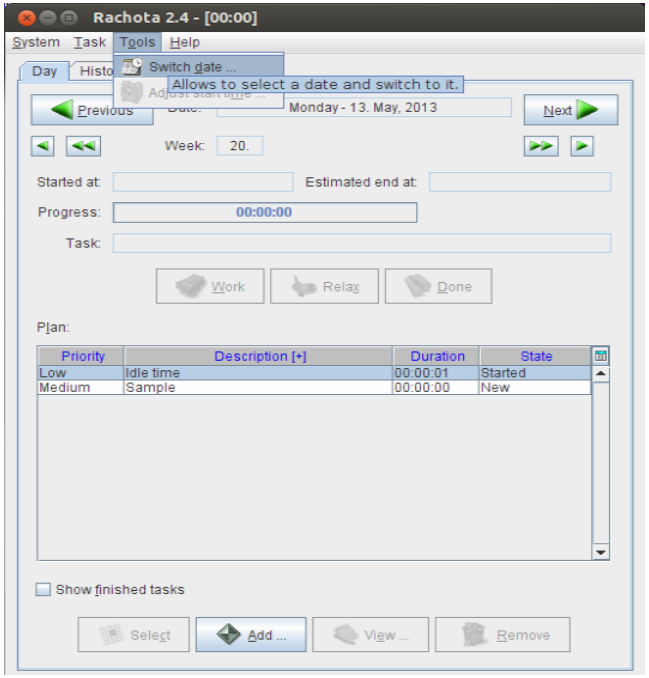
Sources of Variability

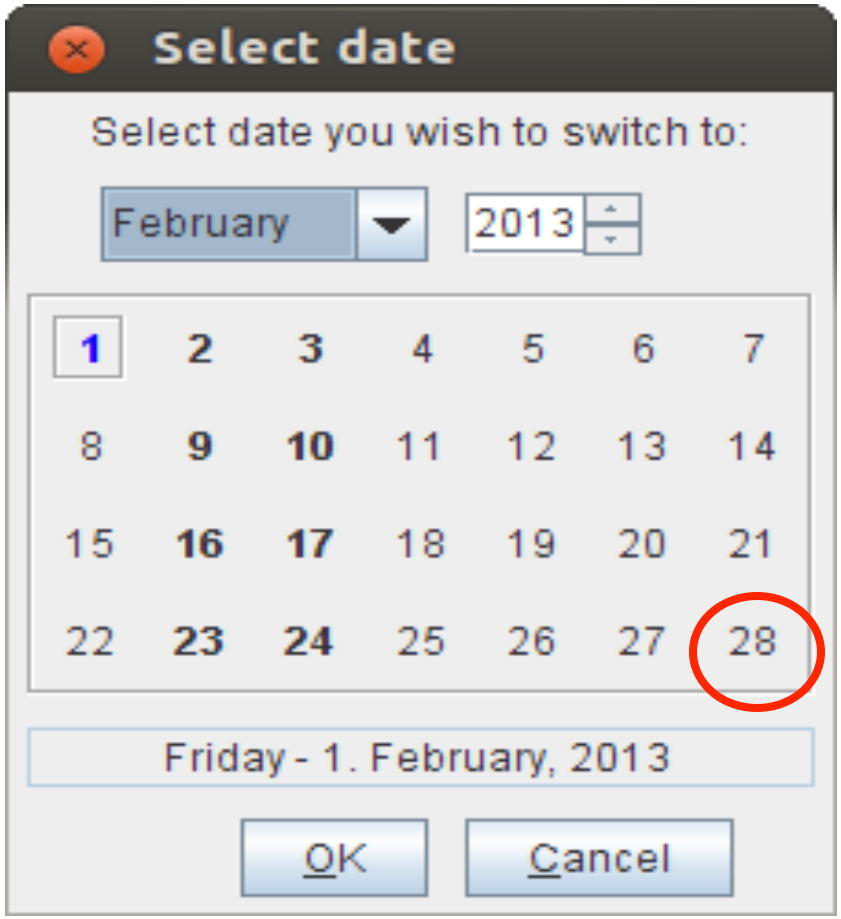
- Testing tool (e.g. GUITAR) options
- Application configuration
- External Data
- OS versions, Java
- System/Environment

Example: External Input Data

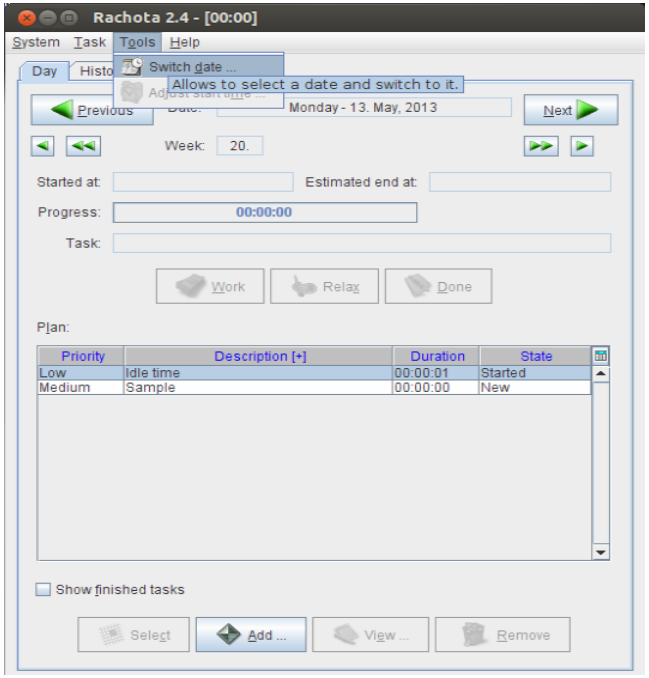


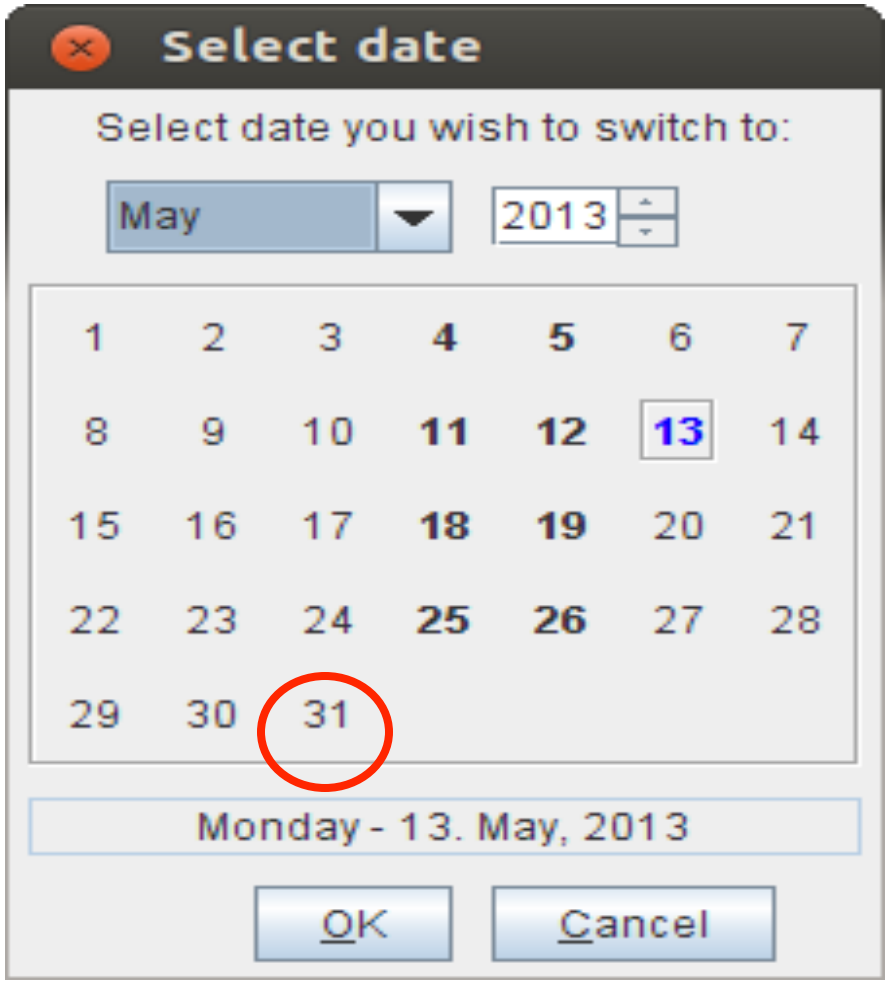
Example: External Input Data



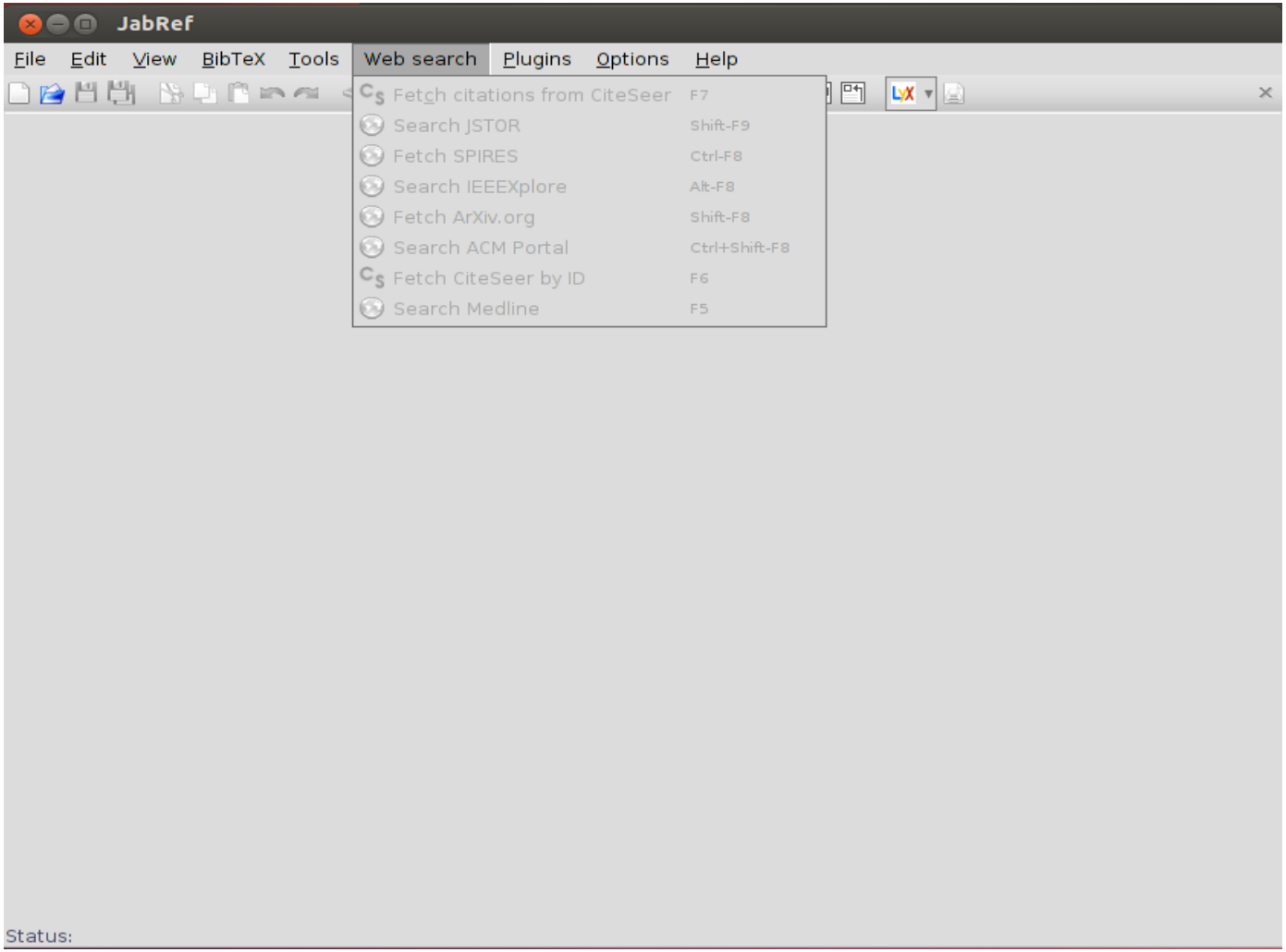


Example: External Input Data

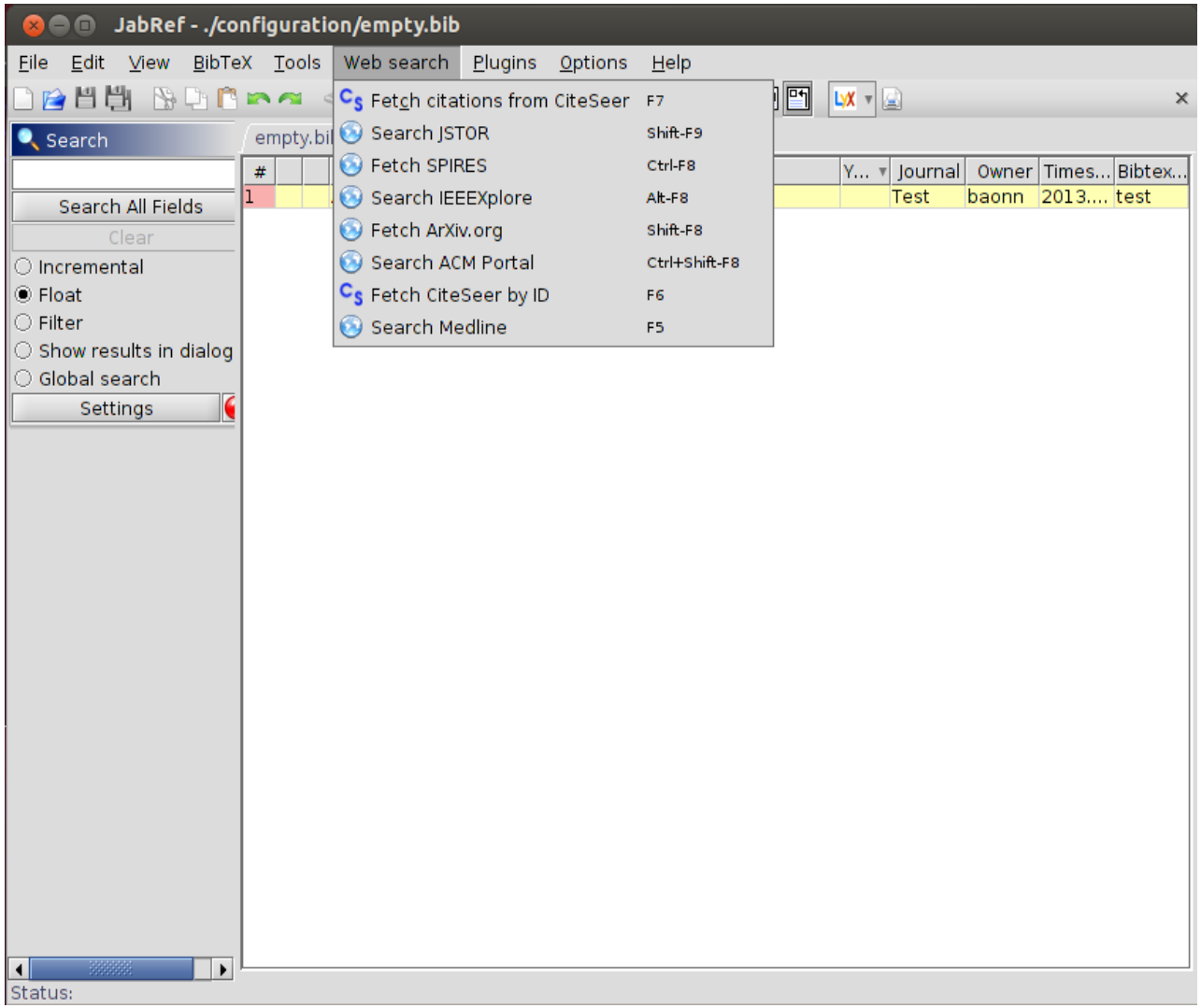




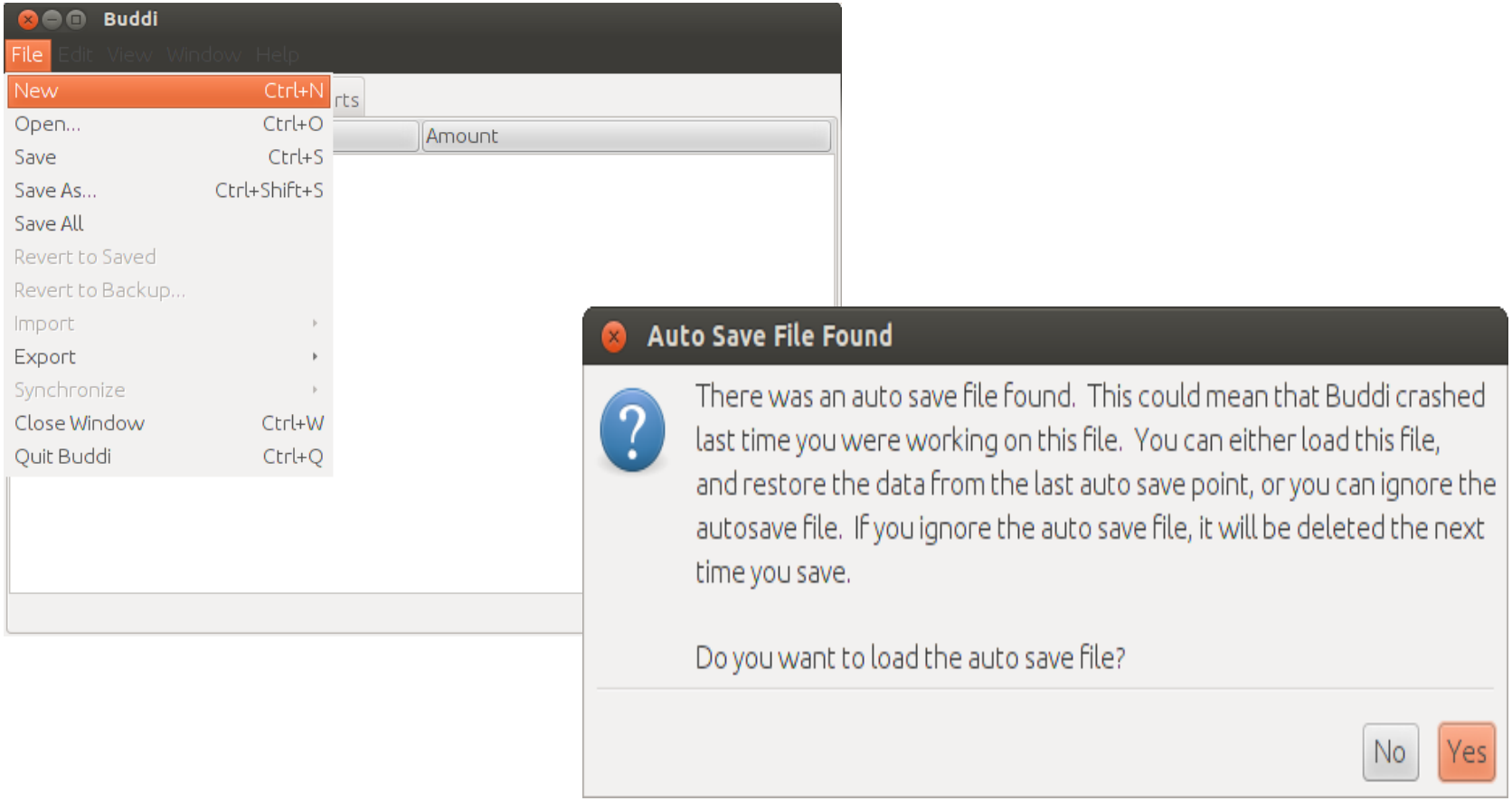
Initial Application State



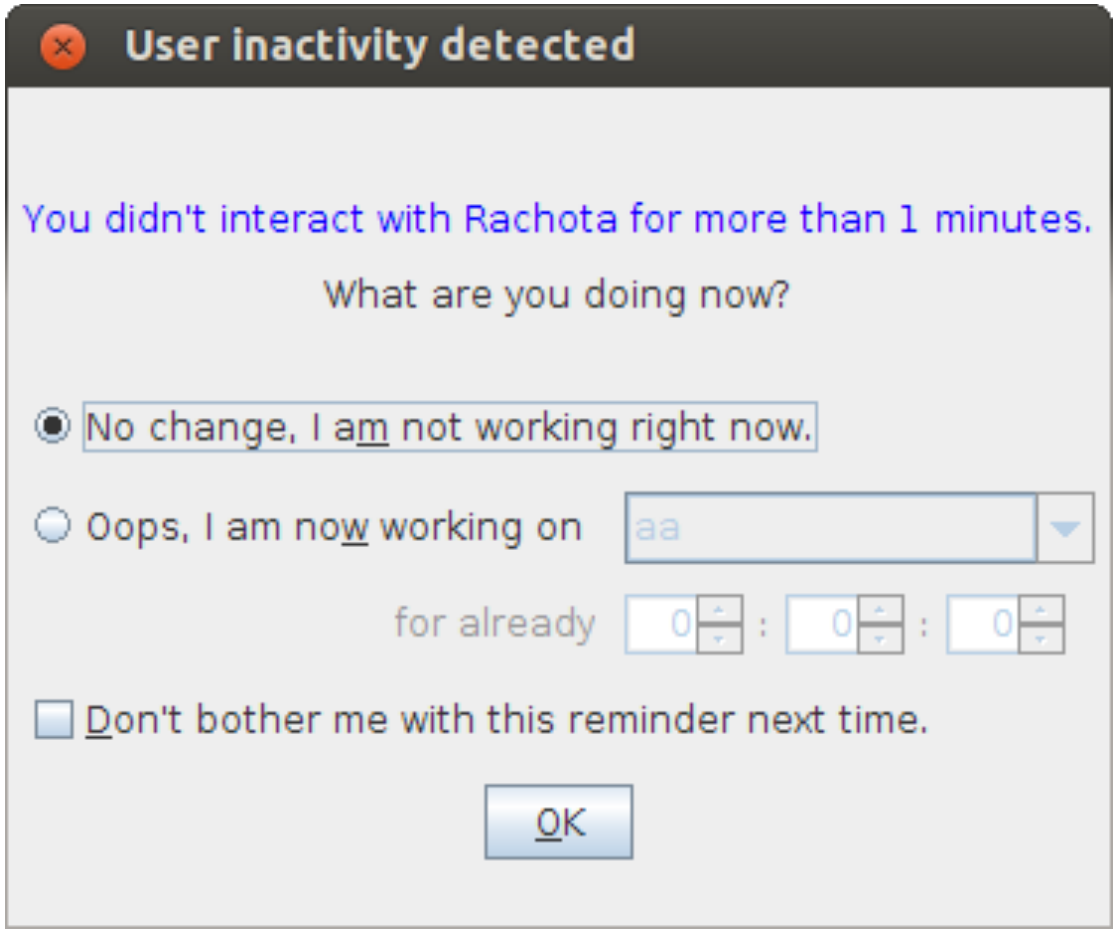
Initial Application State



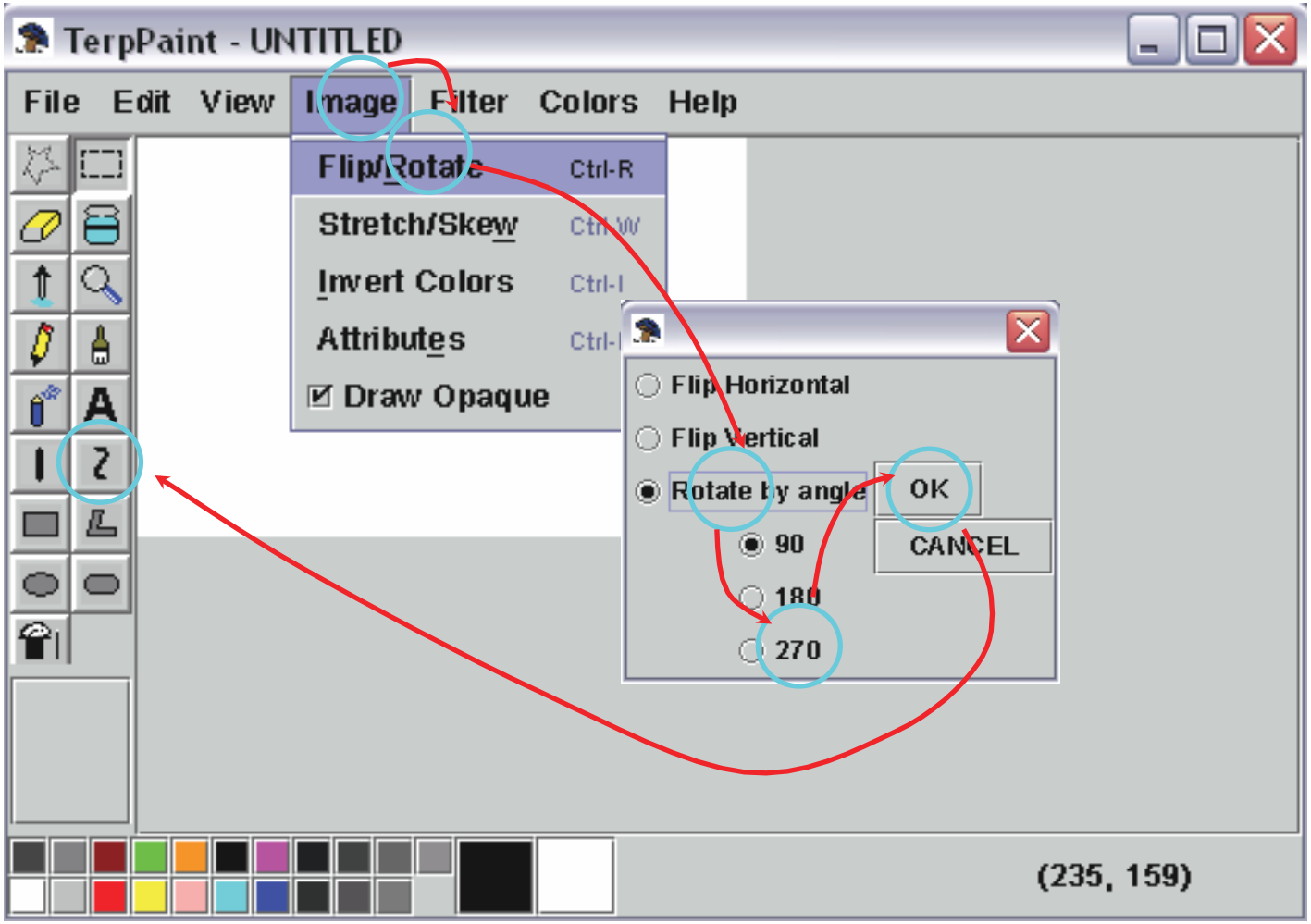
Initial State



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

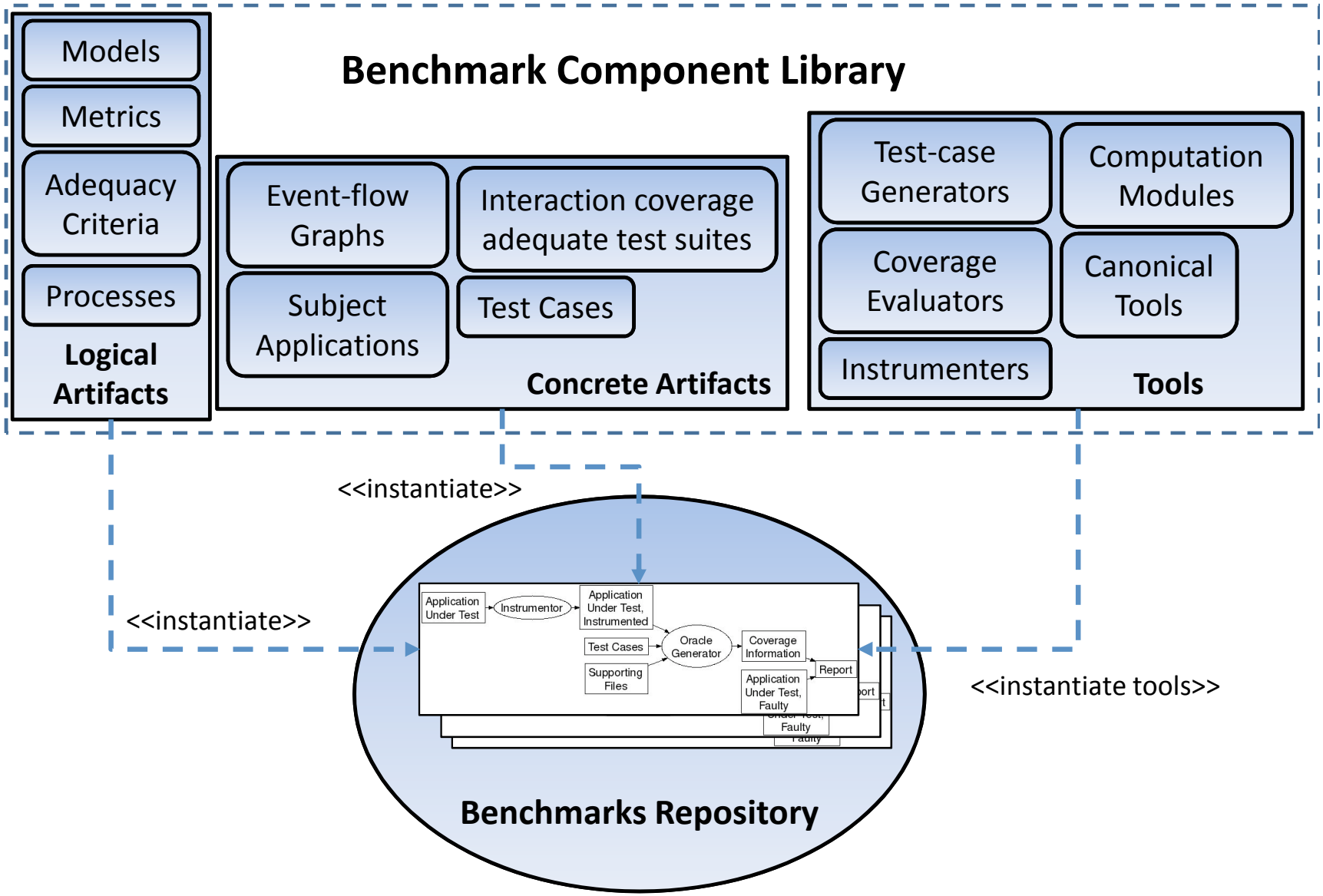
How Can We Reduce Variability?

- 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

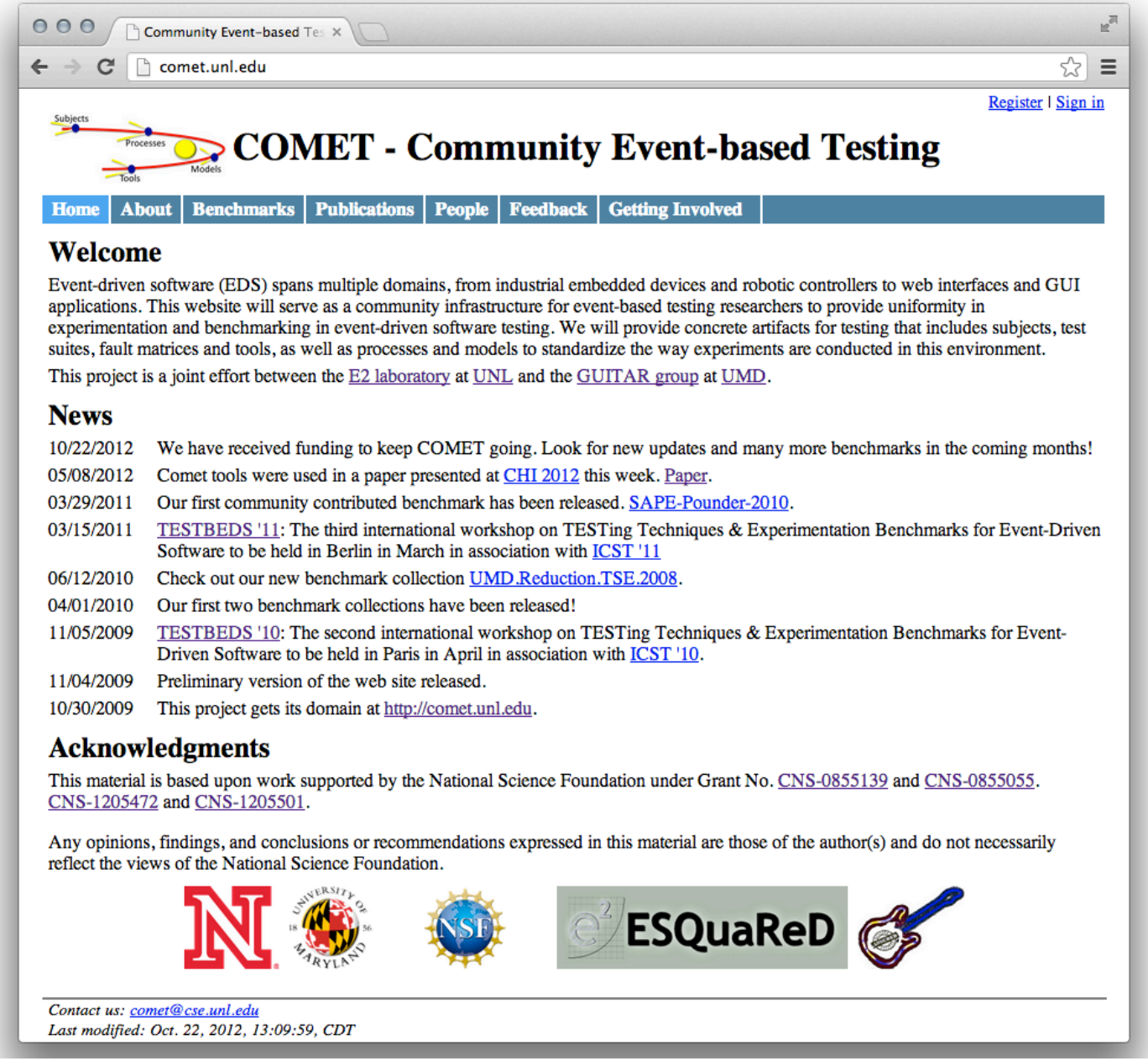
COMET

- **COM**munity **E**vent-based **T**esting (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 variaiblity 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



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.

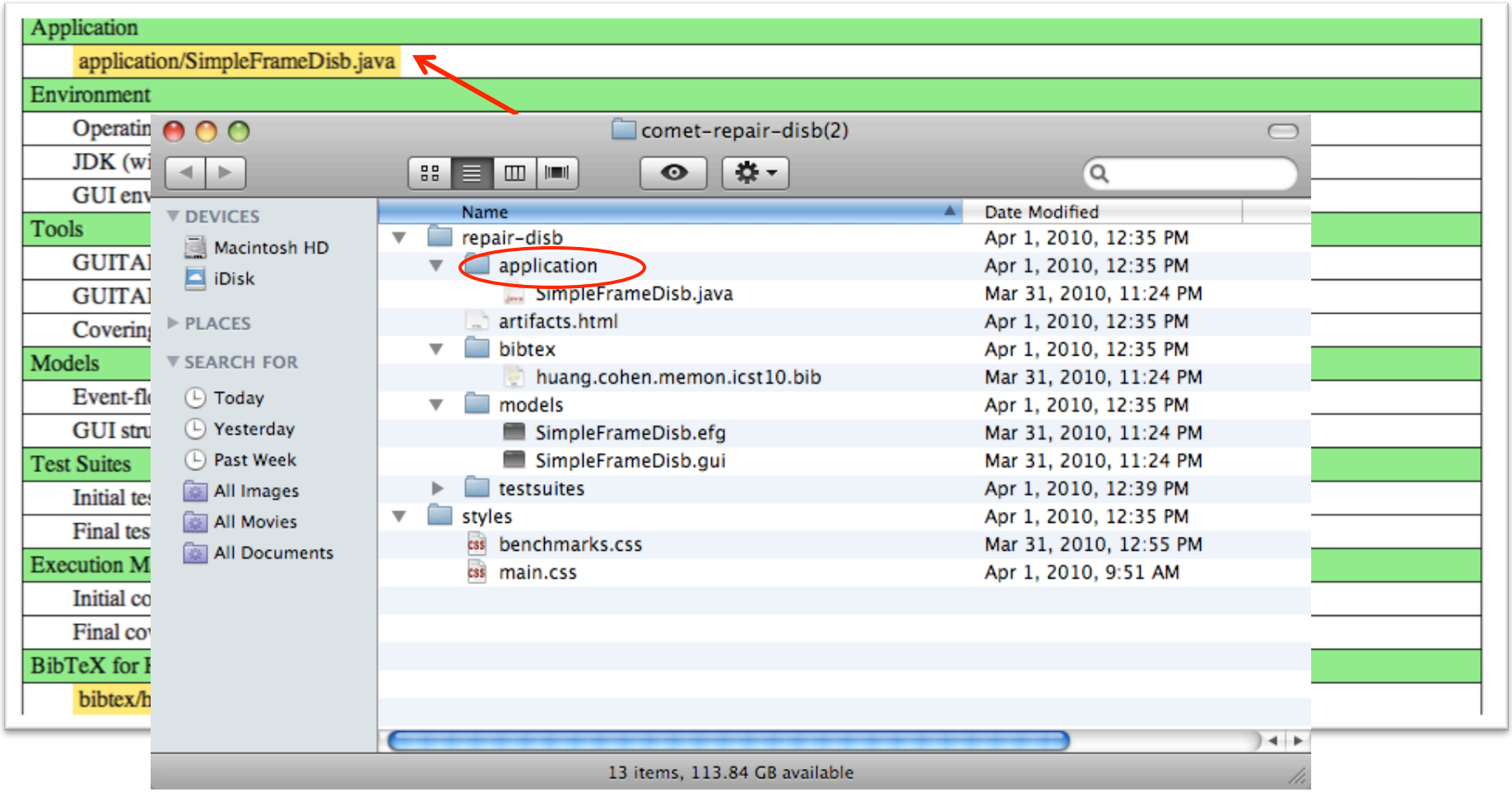
Views: Collections | Benchmarks

	Collection Name	Description

☐	UNL.Toy.2010	This collection of benchmarks was used in the experiments for GUI test suite repair in the paper *Repairing GUI Test Suites Using a Genetic Algorithm*, 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.
		`@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}`
		`}`

☐	UMD.Reduction.TSE.2008	This collection of benchmarks was used in the paper *McMaster, S. and Memon, A. M. Call stack coverage for GUI test-suite reduction. IEEE Trans. Softw. Eng., 2008, IEEE Press.*. The users can use those benchmarks to evaluate different testsuite reduction techniques.
		`@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}`
		`}`

Benchmark Structure

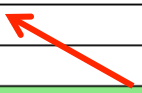


Benchmark Structure

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

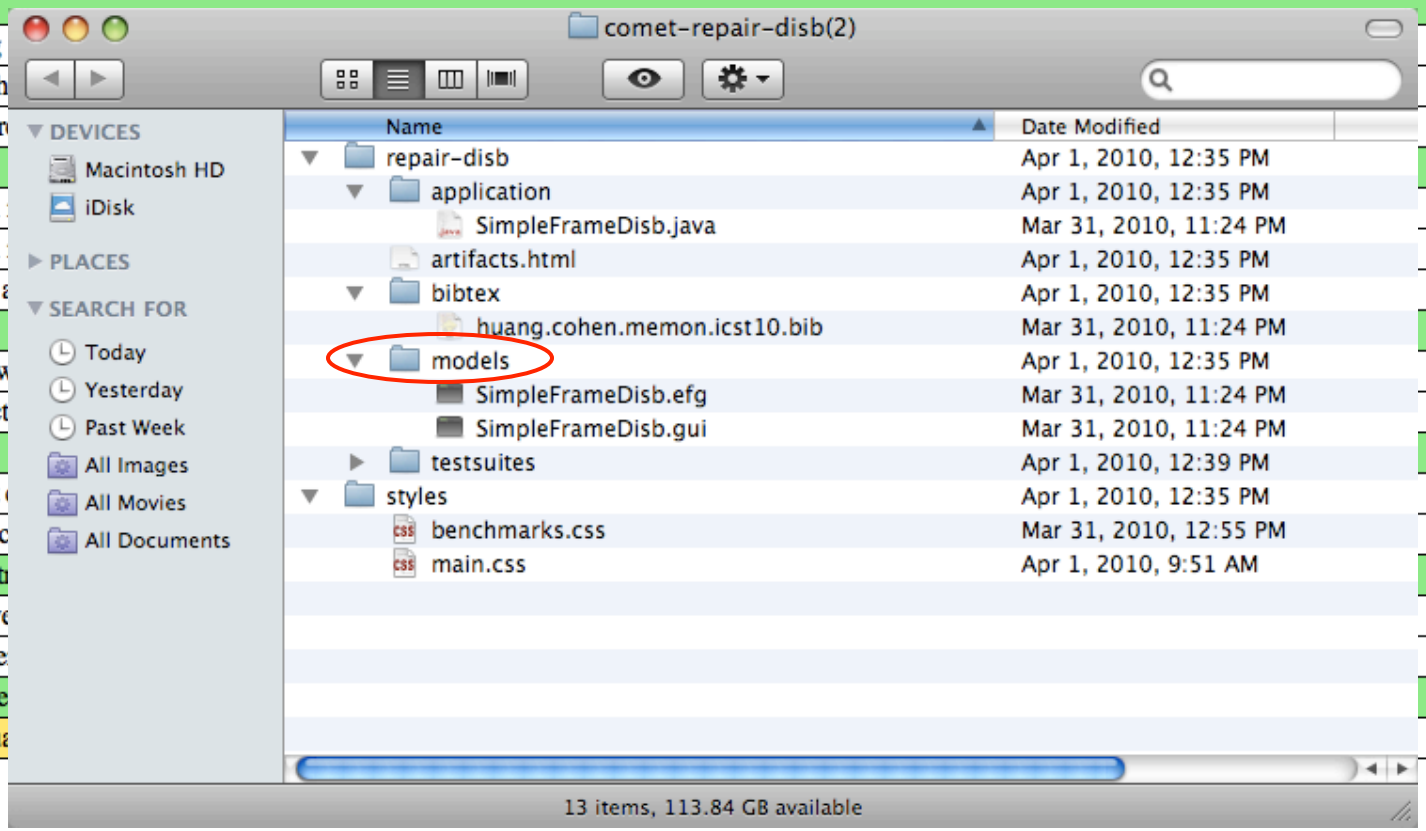
Benchmark Structure

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



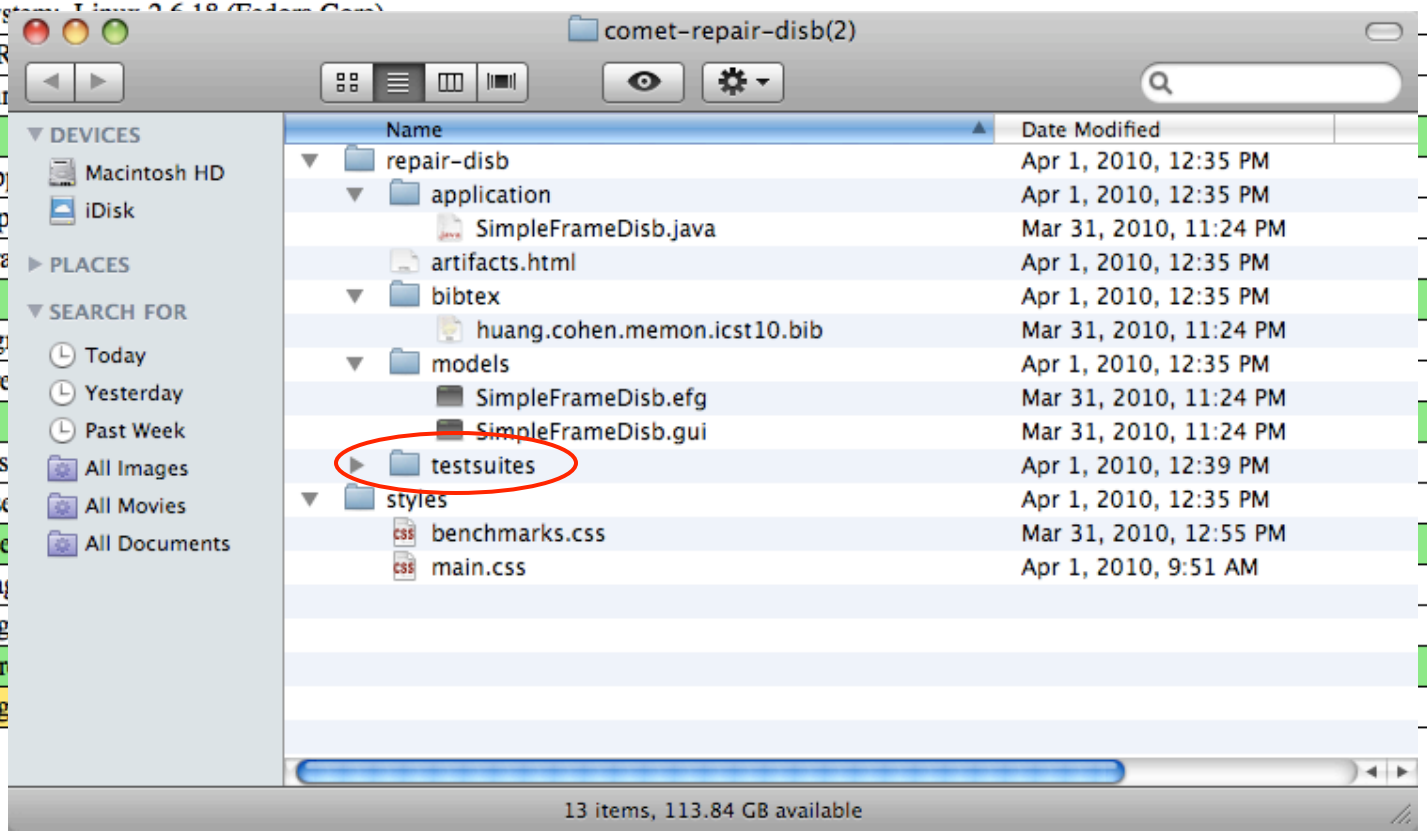
Benchmark Structure

Application
application/SimpleFrameDisb.java
Environment
Operating
JDK (with
GUI envir
Tools
GUITAR
GUITAR
Covering a
Models
Event-flow
GUI struct
Test Suites
Initial test
Final test c
Execution Mat
Initial cove
Final cove
BibTeX for Re
bibtex/hua



Benchmark Structure

Application
application/SimpleFrameDisb.java
Environment
Operating sys
JDK (with JR
GUI environ
Tools
GUITAR rip
GUITAR rep
Covering arra
Models
Event-flow gr
GUI structure
Test Suites
Initial test cas
Final test case
Execution Matrice
Initial coverag
Final coverag
BibTeX for Refer
bibtex/huang



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

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