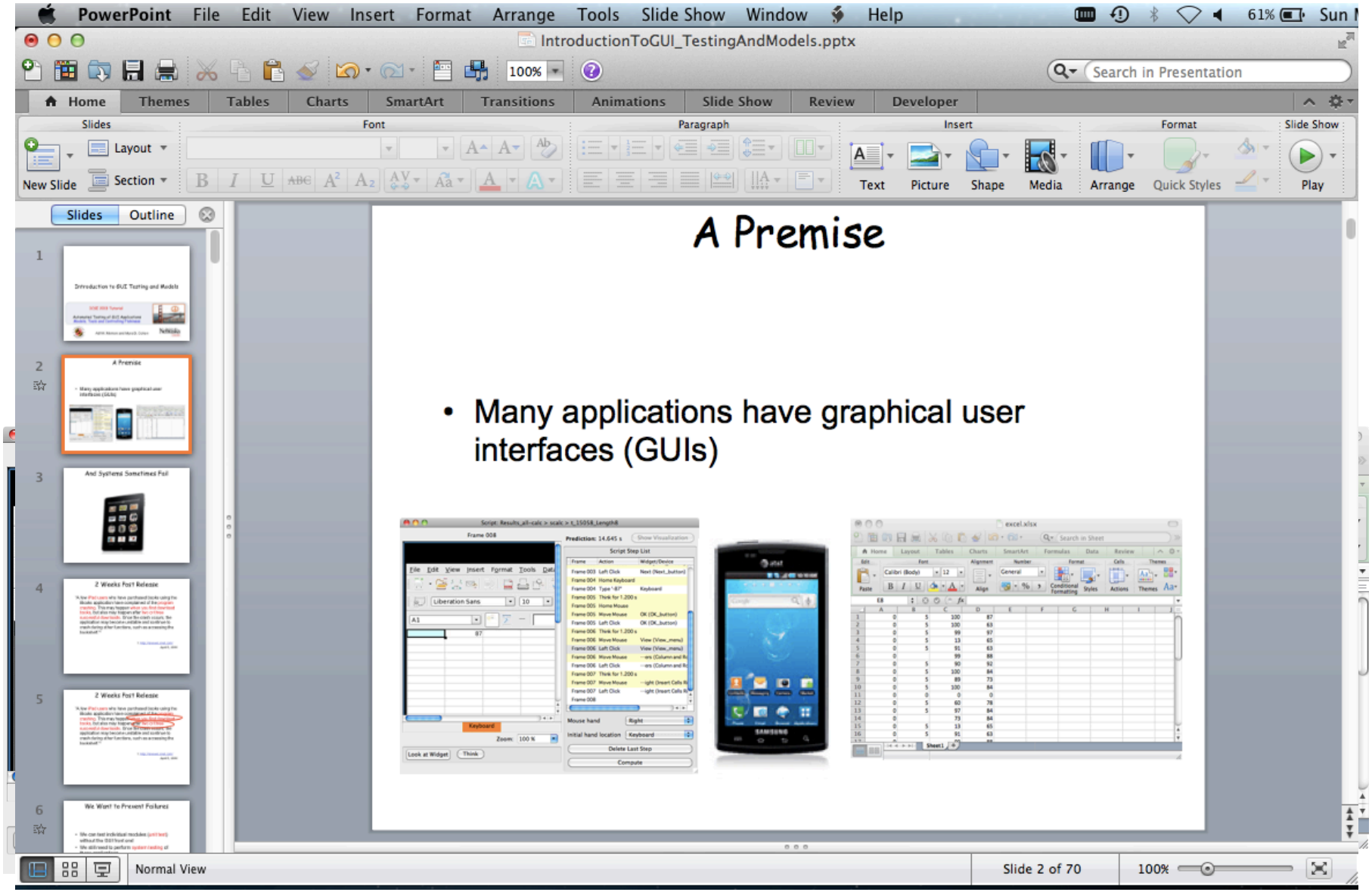
Introduction to GUI Testing and Models

ICSE 2013 Tutorial

Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Atif M. Memon and Myra B. Cohen

A Premise


PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
IntroductionToGUI_TestingAndModels.pptx
Search in Presentation
Home Themes Tables Charts SmartArt Transitions Animations Slide Show Review Developer
Slides Font Paragraph Insert Format Slide Show
New Slide Layout Section Text Picture Shape Media Arrange Quick Styles Play
Slides Outline
A Premise
• Many applications have graphical user interfaces (GUIs)
Normal View
Slide 2 of 70
100%

And Systems Sometimes Fail

2 Weeks Post Release

"A few iPad users who have purchased books using the iBooks application have complained of the program crashing. This may happen when you first download books, but also may happen after two or three successful downloads. Once the crash occurs, the application may become unstable and continue to crash during other functions, such as accessing the bookshelf."[1]

1. http://reviews.cnet.com/

April 5, 2010

2 Weeks Post Release

"A few iPad users who have purchased books using the iBooks application have complained of the program crashing. This may happen when you first download books, but also may happen after two or three successful downloads. Once the crash occurs, the application may become unstable and continue to crash during other functions, such as accessing the bookshelf."[1]

1. http://reviews.cnet.com/

April 5, 2010

We Want to Prevent Failures

- We can test individual modules (unit test) without the GUI front end
- We still need to perform system testing of these applications
  - Performed through the GUI
  - Test cases are sequences of events (e.g. click/ touch button or widget, rotate)

GUI System Testing Can Be Effective

- GUI testing can uncover faults in:
  - GUI code
  - Glue code
  - Underlying business logic

[Brooks, Robinson, Memon, An Initial Characterization of Industrial Graphical User Interface Systems" ICST 2009]

But it is Challenging

- Behavioral/interface specifications are often not well defined
- Have an enormous potential event-space
- Platform specificity
- Need automation

Automated GUI Testing

- Random actions on the GUI

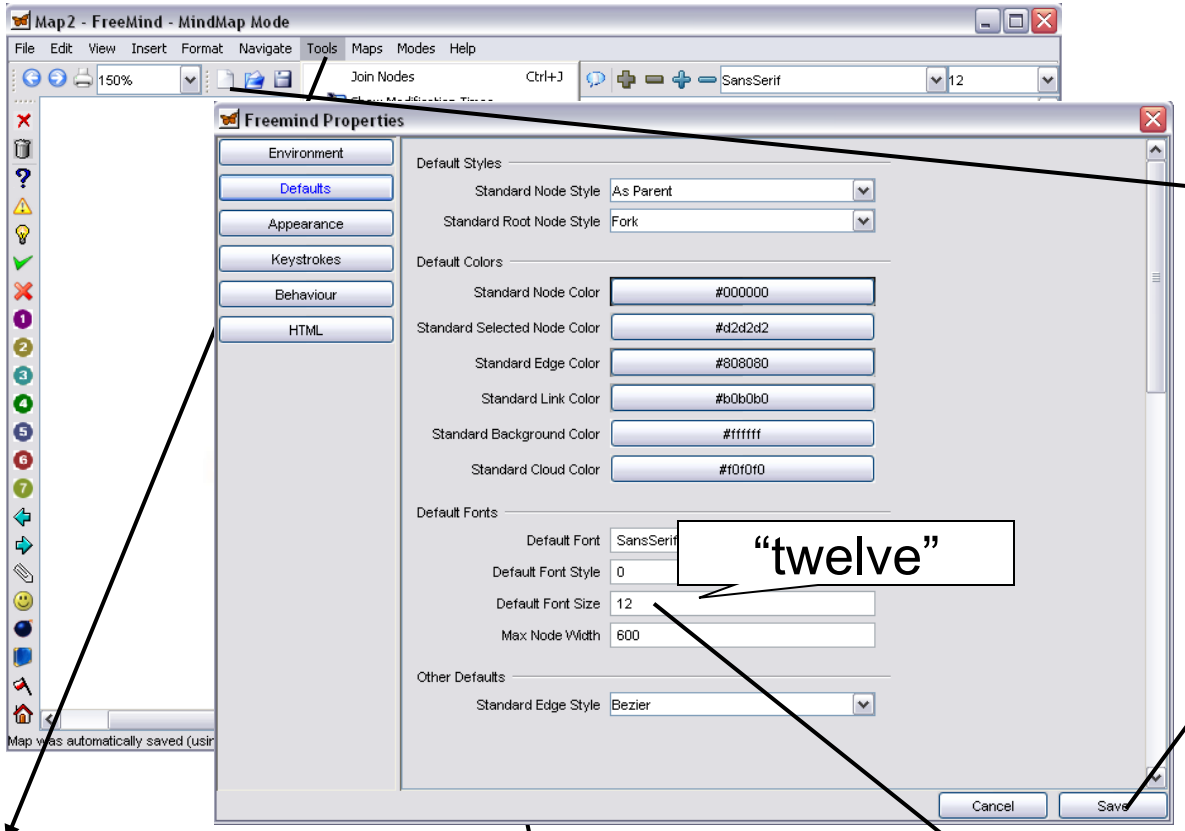
Or:

- Create a model of the GUI
  - State machine models
  - Graph model (abstracts out some behavior)

In This Lecture

- We will present an overview of model based GUI Testing

Typical System GUI Test



Crash

NewMapAction.
actionPerformed

e_4:Save

Properties.save
ButtonClicked

e_1: Tools

StructuredMenuHolder.
actionPerformed

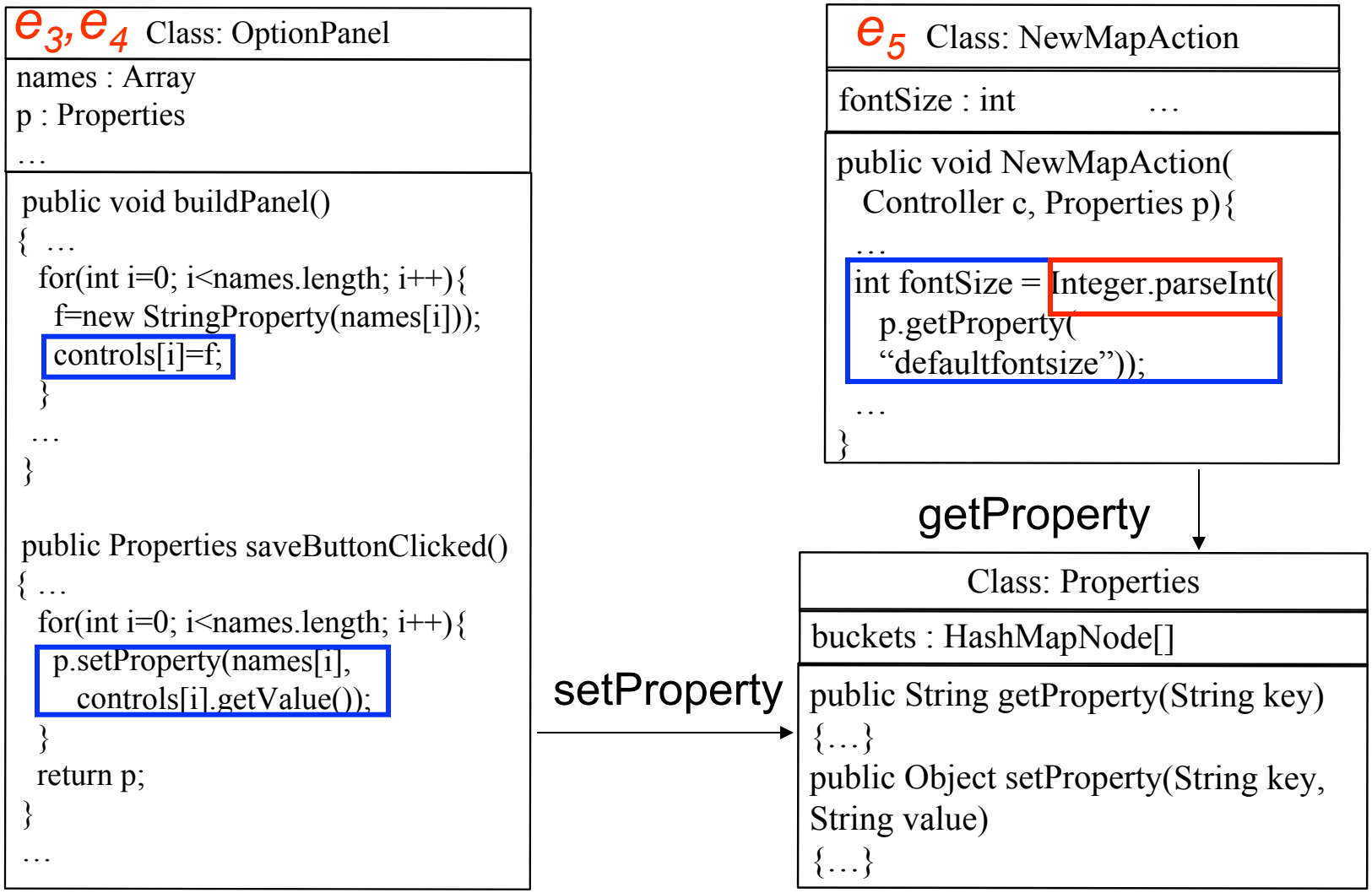
e_2:Preferences…

OptionPanel.
buildPanel

e_3:Default Font Size

PropertyControl.
actionPerformed

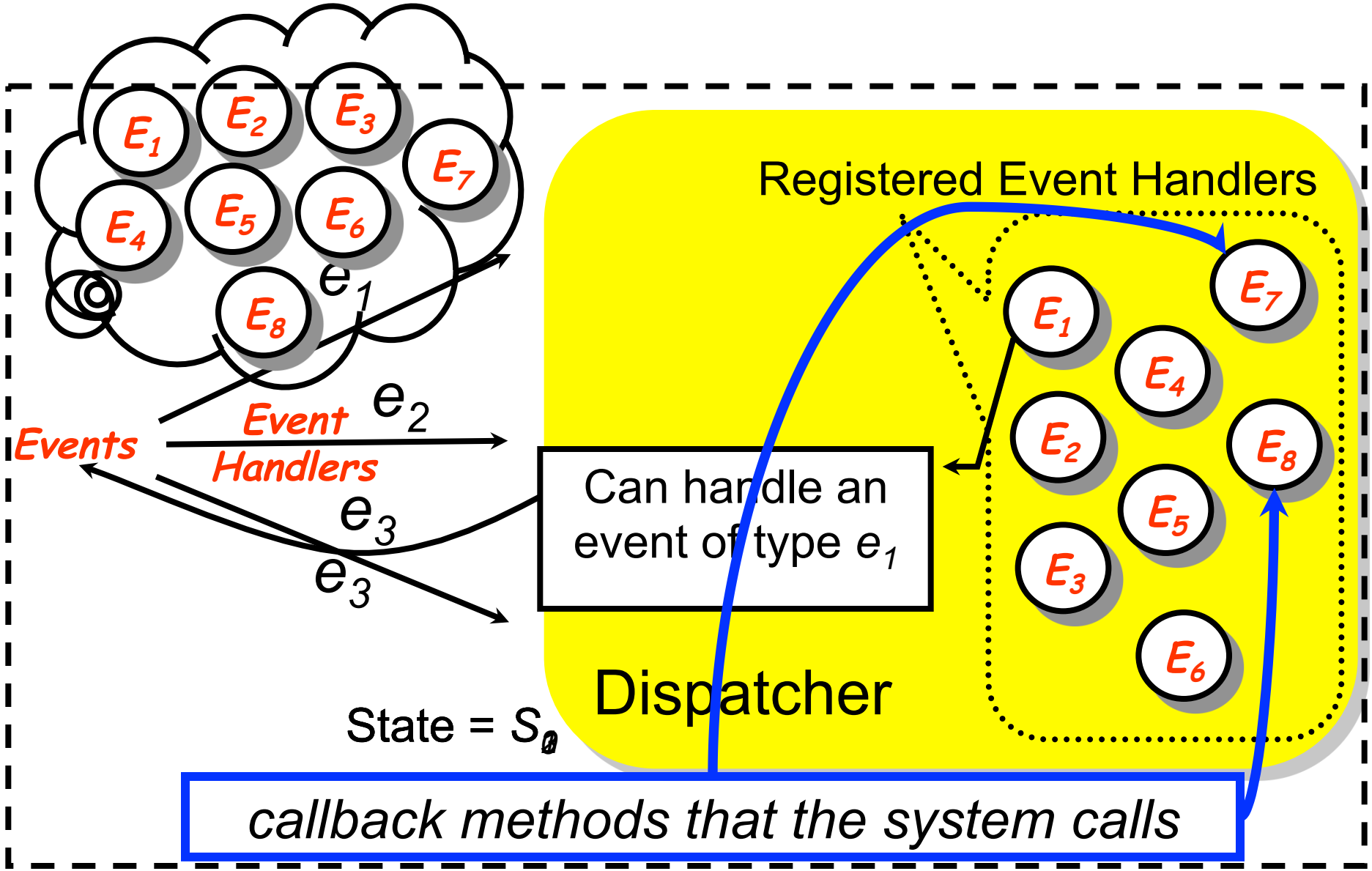
Associated Code



Complexity of GUI Testing

- GUI faults surface with precise event sequences
  - Drive the software into particular states
- Multiple components/classes involved
  - Static analysis does not scale
- Source for all parts of GUI not always available
  - Library code
- GUI is “big integrator” of software
  - It brings together all code together
    - Many times these code pieces never tested together

GUI Software in Action!

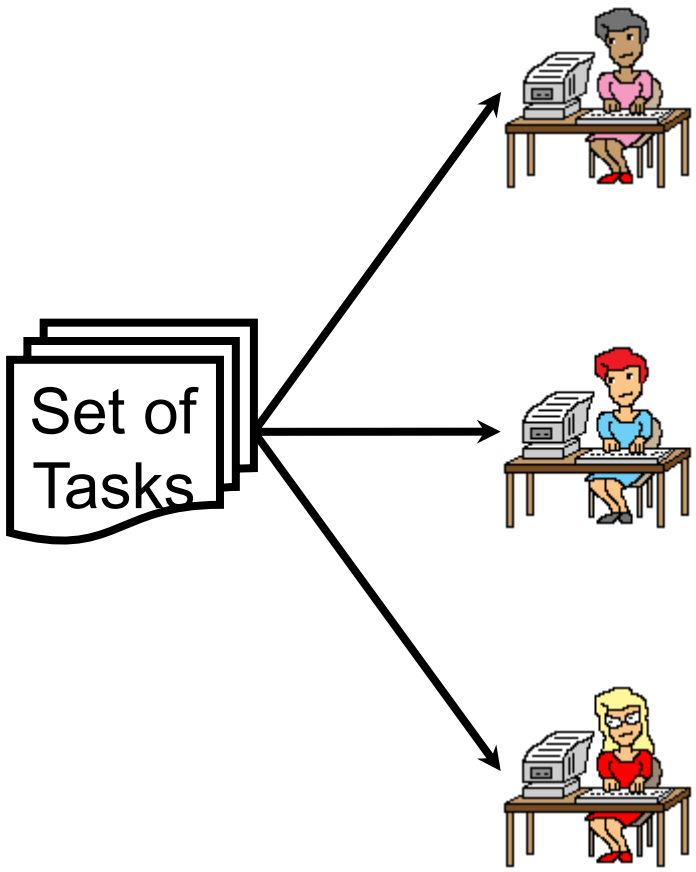


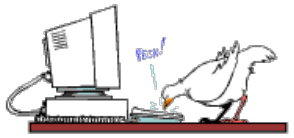
GUI Testing is Complex!

- Hard to do system testing
  - multi-language, reflection, callbacks and multi-threading, etc.

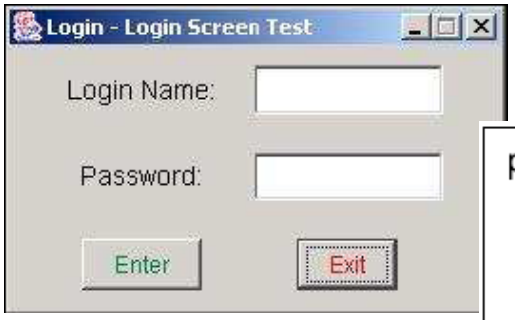
- GUIs impose work-flows over components (event handlers)
- GUI event sequences trigger component interactions
- GUI testing = integration testing of event handlers
- GUI test case = sequence of events
- Huge input and state space

How GUIs are Tested - Manual



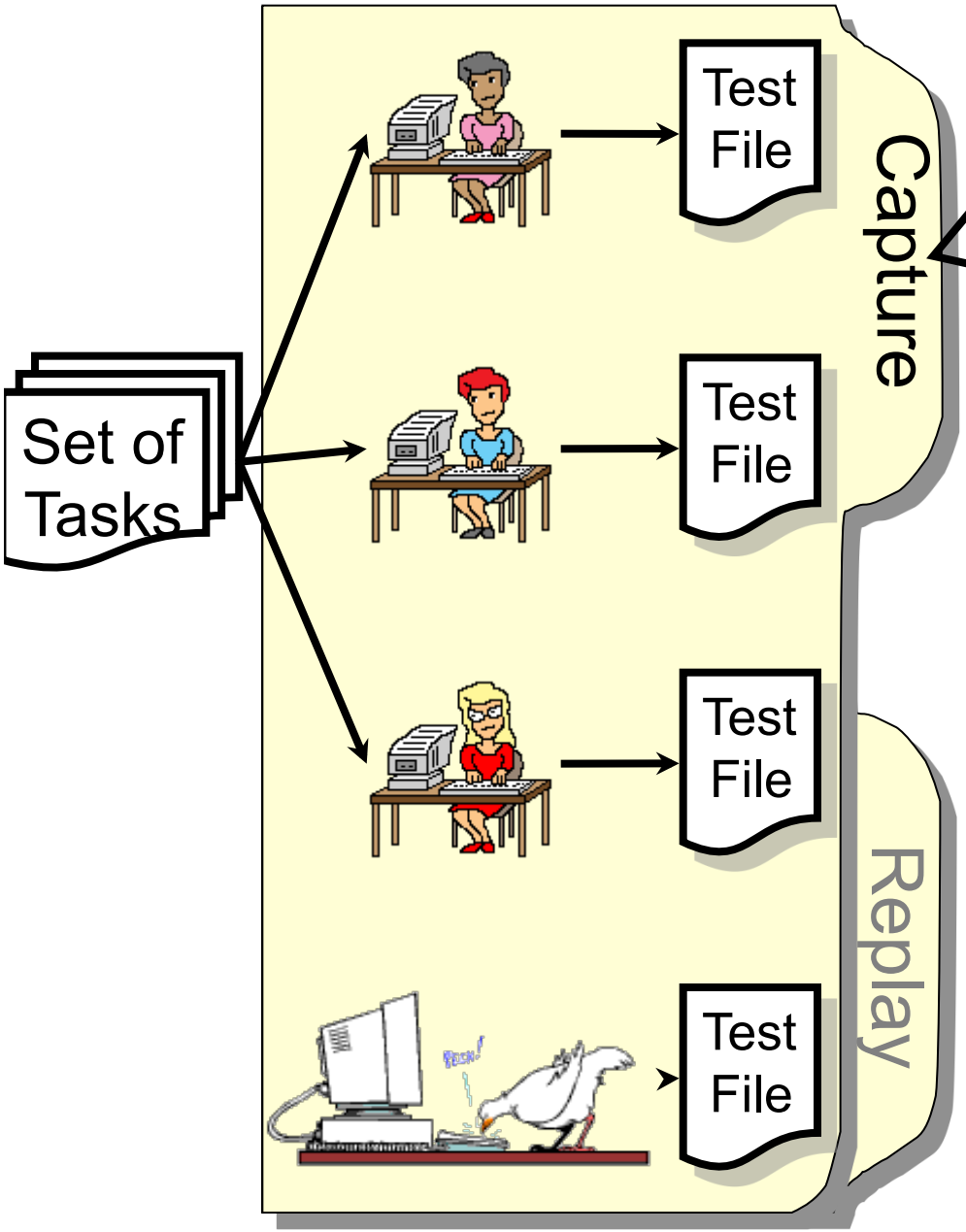


How GUIs are Tested – Code Tests



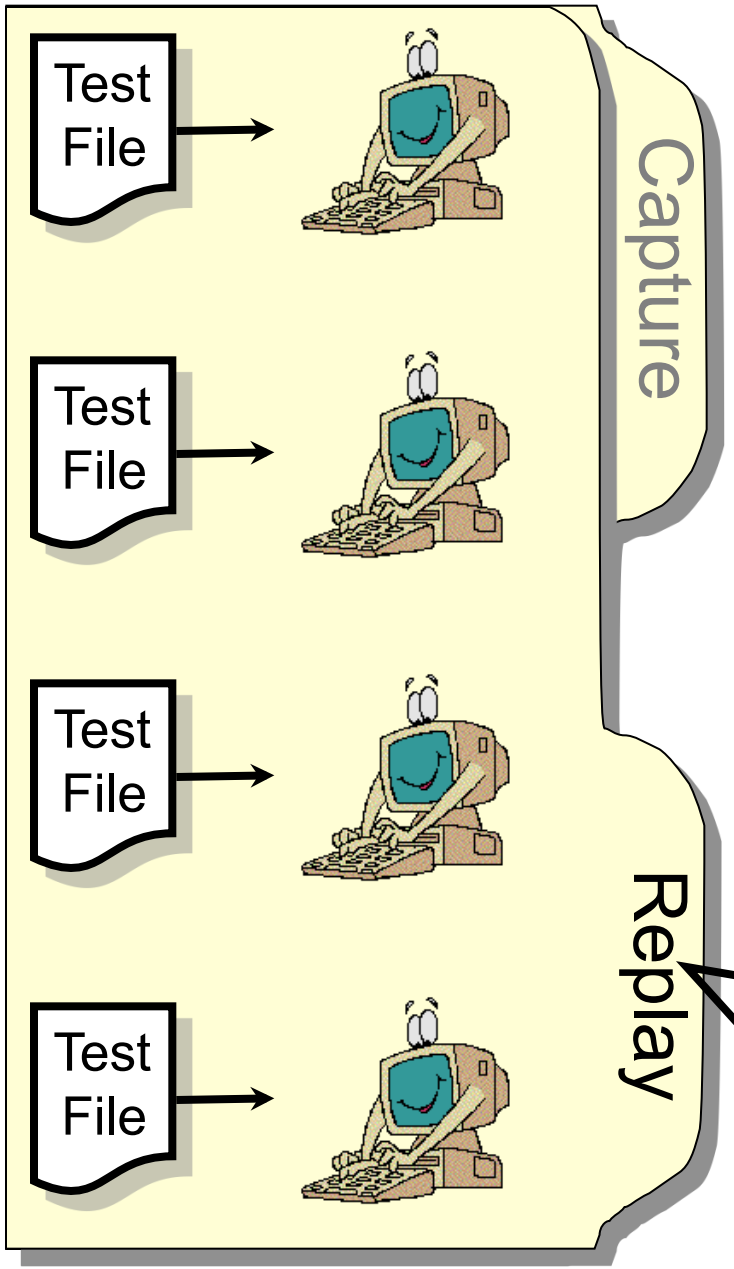

```
public void testLoginScreen() {
    ...

    // (1) type in "qing" into the login name textbox, and "whatever" to the password textbox
    getHelper().sendString( new StringEventData( this, userNameField, "qing" ) );
    getHelper().sendString( new StringEventData( this, passwordField, "whatever" ) );

    // (2) click on "Enter" button
    getHelper().enterCickAndLeave( new MouseEventData( this, enterButton ) );

    // (3) waiting for response
    DialogFinder dFinder = new DialogFinder(null);
    dFinder.setWait(0);

    // (4) login screen window is disposed
    showingDialogs = dFinder.findAll( loginScreen );
    assertEquals( "Number of dialogs showing is wrong", 0, showingDialogs.size( ) );
    assertTrue( "Login screen is showing still", !loginScreen.isShowing( ) );

    // (5) main GUI window shows up
    FrameFinder fFinder = new FrameFinder(null);
    showingWindows = fFinder.findAll();
    assertEquals( "Number of windows showing is wrong", 1, showingWindows.size( ) );
     ...
}
```

How GUIs are Tested - Capture/Replay



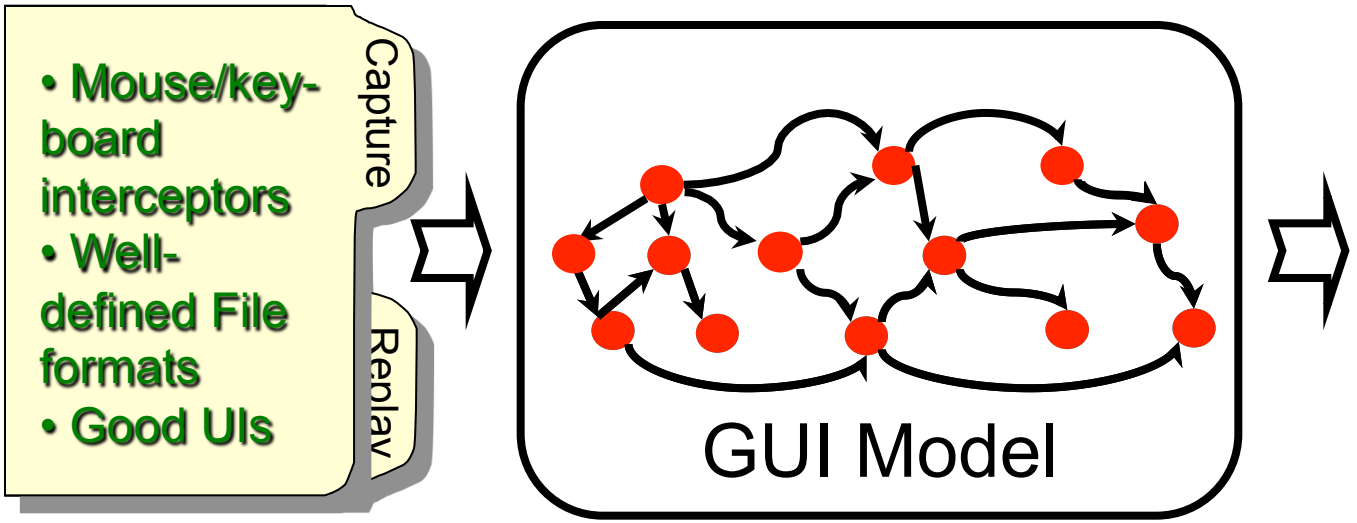
- Tester MANUALLY performs events on the GUI
- Tool records all user inputs
- Partial response/ state as specified by tester

How GUIs are Tested – Capture/Replay



- Tool replays tester's actions
- **Verifies application response against stored response**

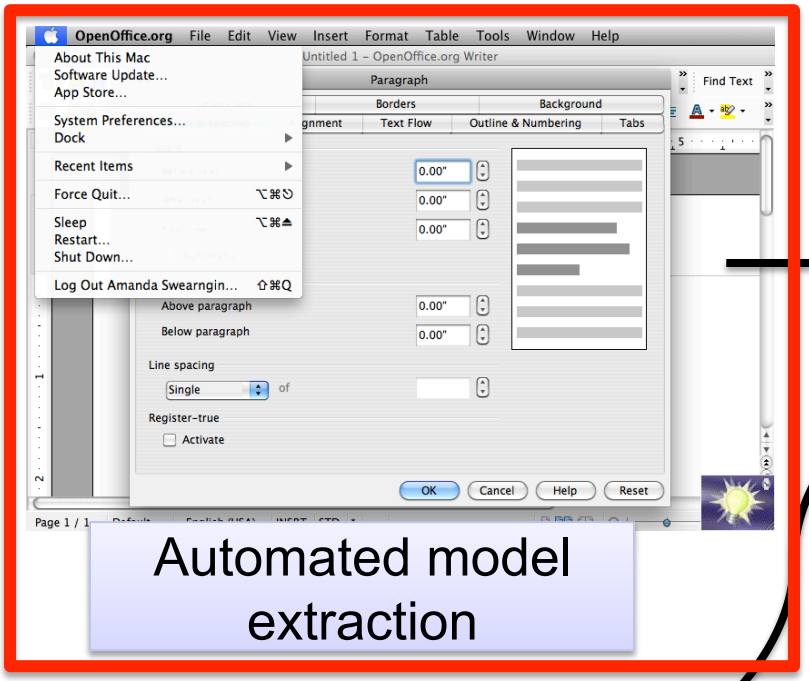
Model-based Testing



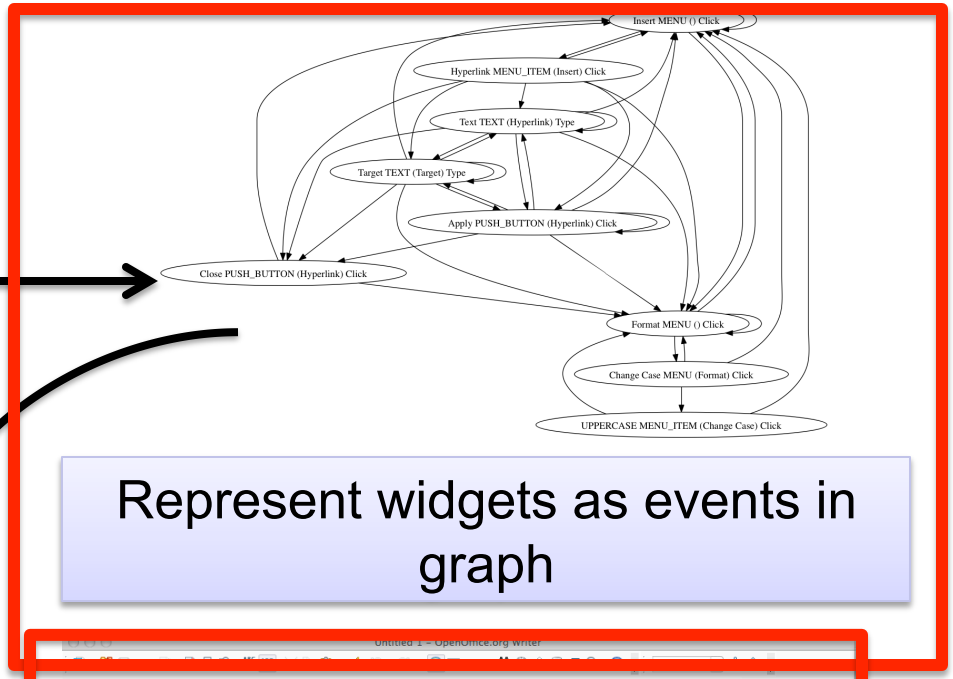
- Start with capture/replay tools
  - Well-engineered
  - Test management

- Pre/postconditions
  - AI planning
- Directed graph models
  - Event-flow graph
  - Event-interaction graph
- Covering arrays
- Genetic algorithms
- Probabilistic state machines
- Run-time behavior

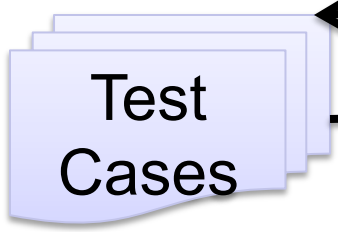
Graph-Based Approach



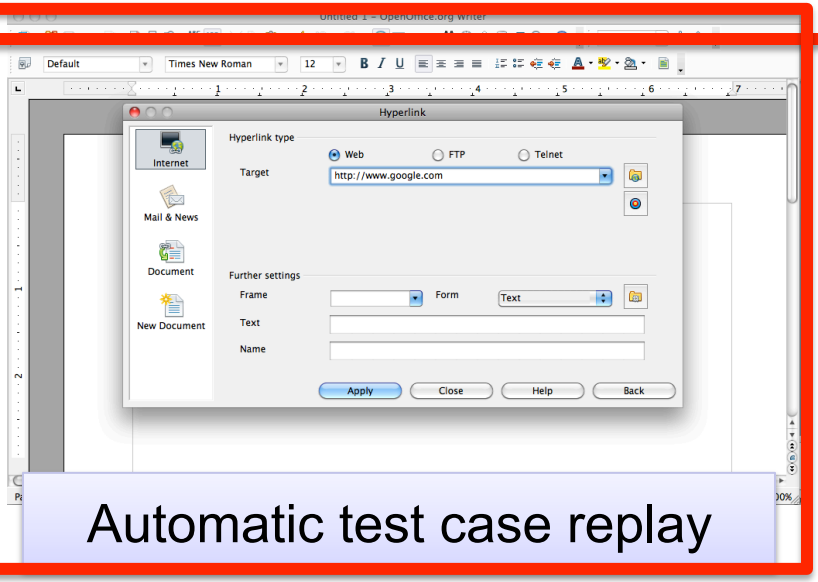
Automated model extraction



Represent widgets as events in graph



Test case generation



Automatic test case replay

Today's Use Case

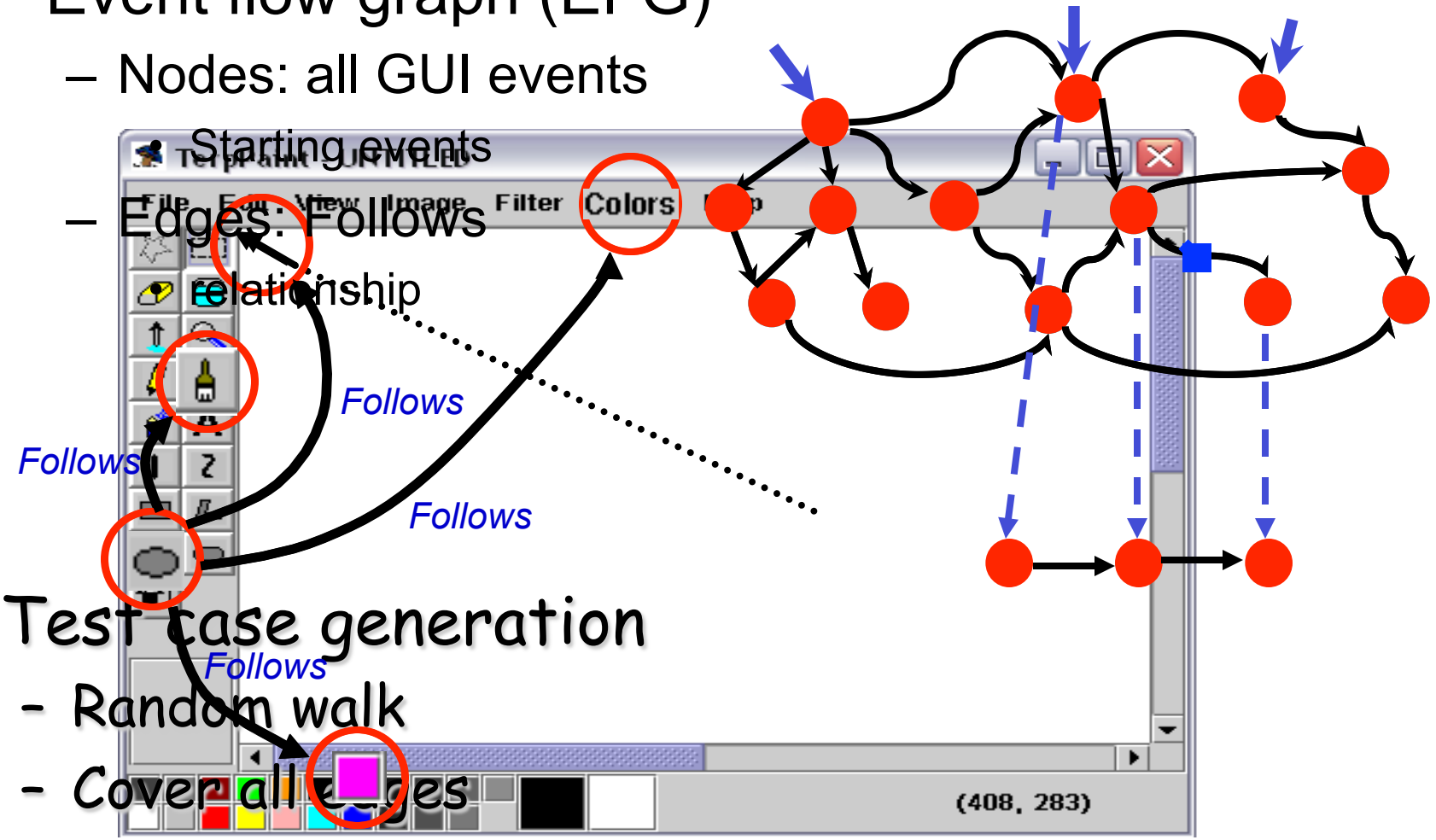
- Given a GUI
  - Don't have a formal spec. or model
  - Can't create a state-machine because it would be too large
- Solution: Adaptive test case generation
  - Fully automatic
  - Create a preliminary approximate model
  - Generate test cases from the model
  - Use the results to "refine" the model
  - Iterate
- Challenges
  - What type of models?
    - How to obtain the preliminary model?
    - What is the nature to the refinements?
  - What is the nature of test cases?
  - What results to use from test cases?
  - How to adapt?

Directed Graph Models

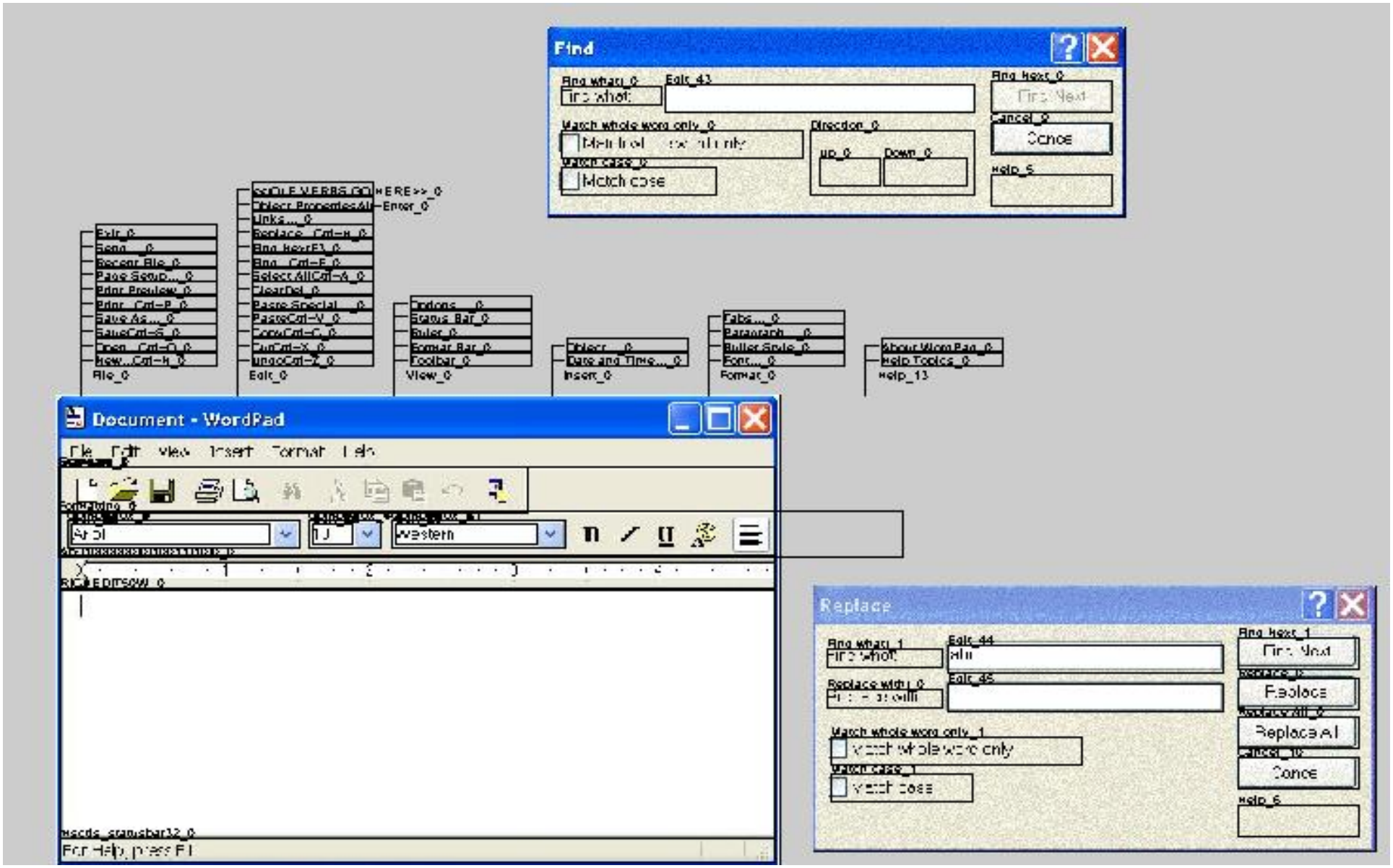
- Model the space of GUI interactions as a graph
  - i.e., given a GUI, create a graph model of all the possible sequences that a user can execute
  - Use the model to generate event sequences

Sampling The Event-Interaction Space

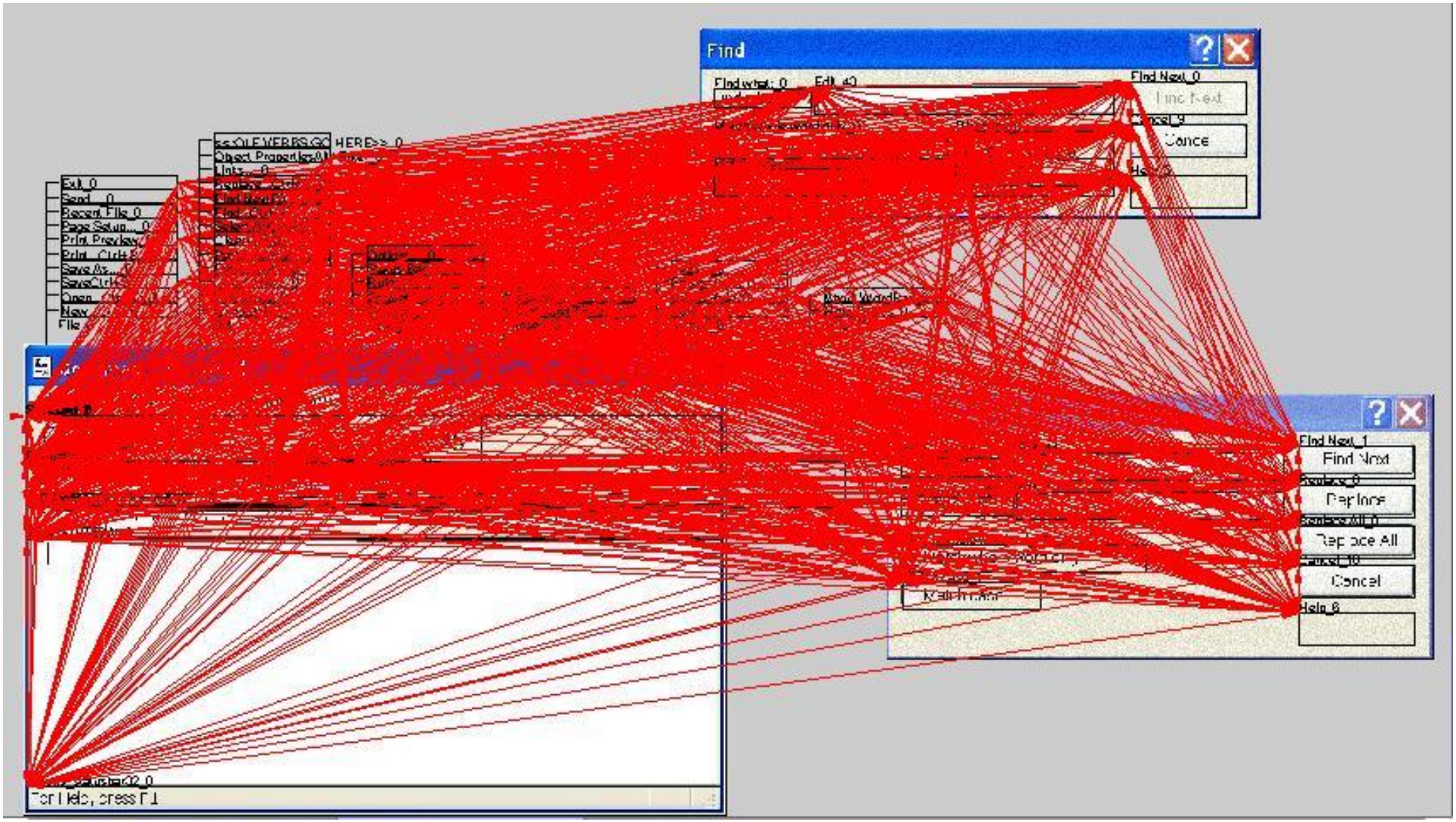
- Event flow graph (EFG)
  - Nodes: all GUI events
    - Starting events
  - Edges: Follows relationship
- Test case generation
  - Random walk
  - Cover all edges



A Part of MS WordPad



Its Event-Flow Graph



Automation

- How do we create the event-flow graph?
  - Create an approximation automatically
    - Reverse Engineering
      - GOAL: Obtain the Event-Flow Graph
      - Fully automatically
      - No source code

Reverse Engineering – GUI Ripping

- Dynamic algorithm
  - No need for source code
- Execute the GUI-based software
  - Traverse the GUI
    - Obtain handle of first window
    - Use windowing API to extract widgets/menus
- Apply transformations
  - Based on GUI dialogs
  - GUI hierarchy
  - Enabled/disabled widgets
- Traverse multiple times if needed

- Engineering Issues
  - Understanding platform-specific GUI frameworks
    - OS-specific GUI handling
  - Introspection
  - Windowing API
- Result – Generic process for GUI Ripping
- MS Windows, Java Swing, Openoffice (UNO - Universal Network Objects), iOS, Android, Web (Selenium), Accessibility API
- Immediate impact – Obtained an approximate EFGs for large GUIs in a few minutes

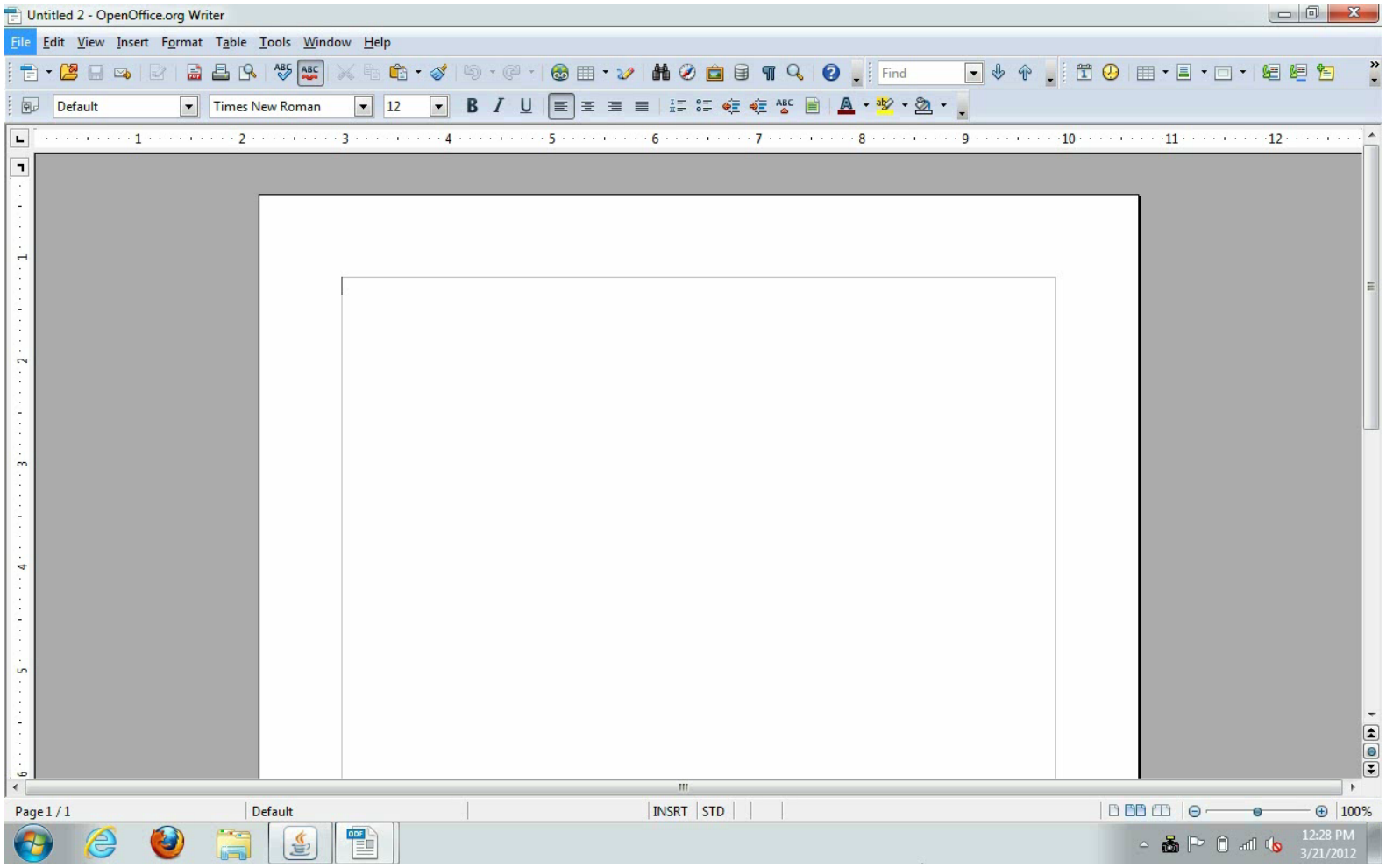
Why is it an Approximate Model?

- GUI Ripper performs one traversal to obtain all event-flows
  - Depth-first (breadth-first for mobile platforms)
  - Miss events/windows
  - Exit application prematurely
  - State-based relationships
    - Conservative algorithm
    - Some event-sequences might be un-executable
- IF we want a better approximation
  - Bootstrap the Ripper
    - Ripper traversal customization
    - Iterative process

Impact of GUI Ripper

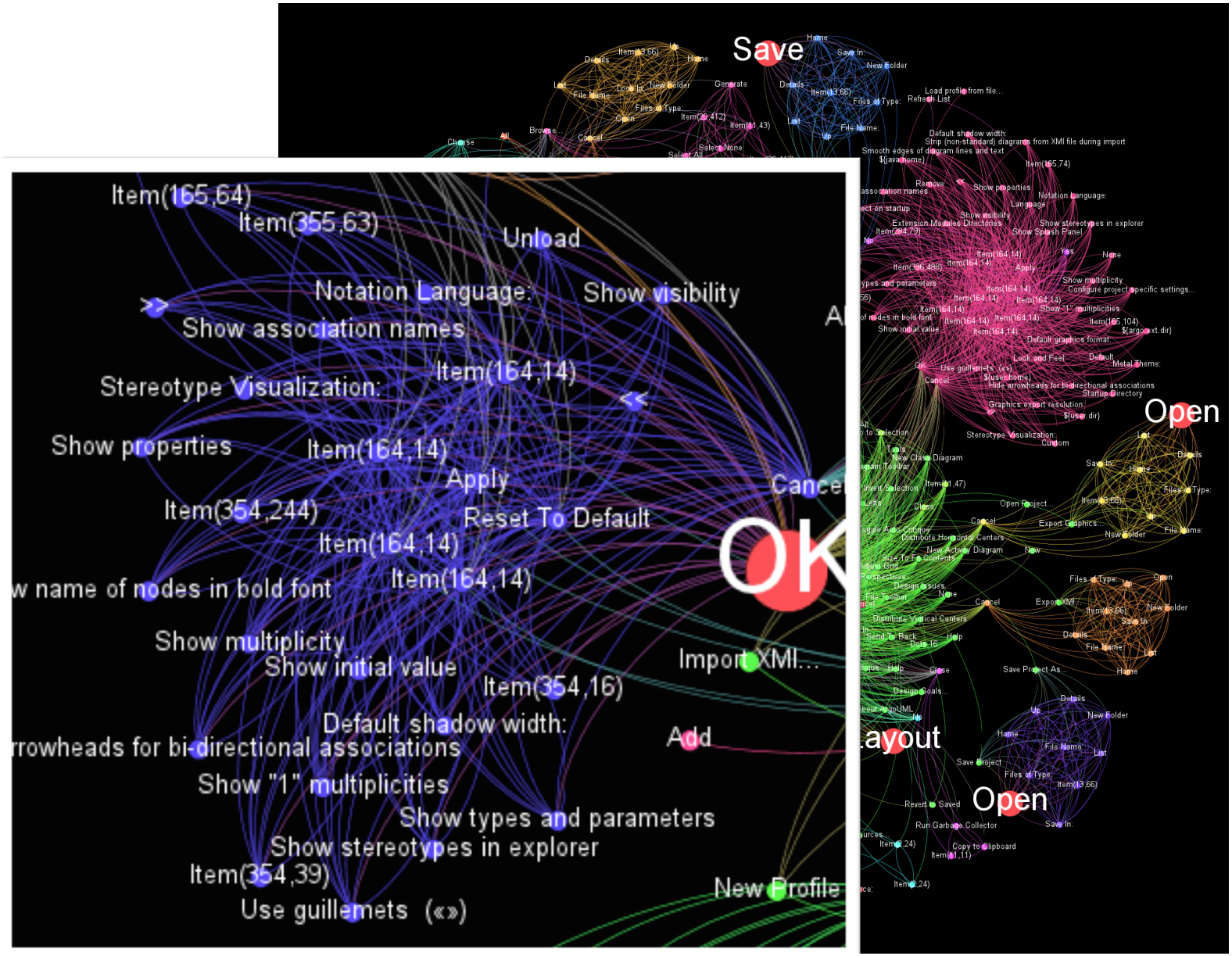
- A way to generate test cases for large GUIs
  - Random, Event-flow graph edge adequate, Code-coverage adequate, Covering arrays
  - Detected several faults in fielded software
- Millions of test cases
  - Dedicated 120 dual-CPU machine cluster
  - CONDOR jobs on UMIACS clusters (1000+ machines)
- For us, this enabled experimentation
  - Examine execution results to better understand the nature of GUI software

Extracting (Ripping) the Model

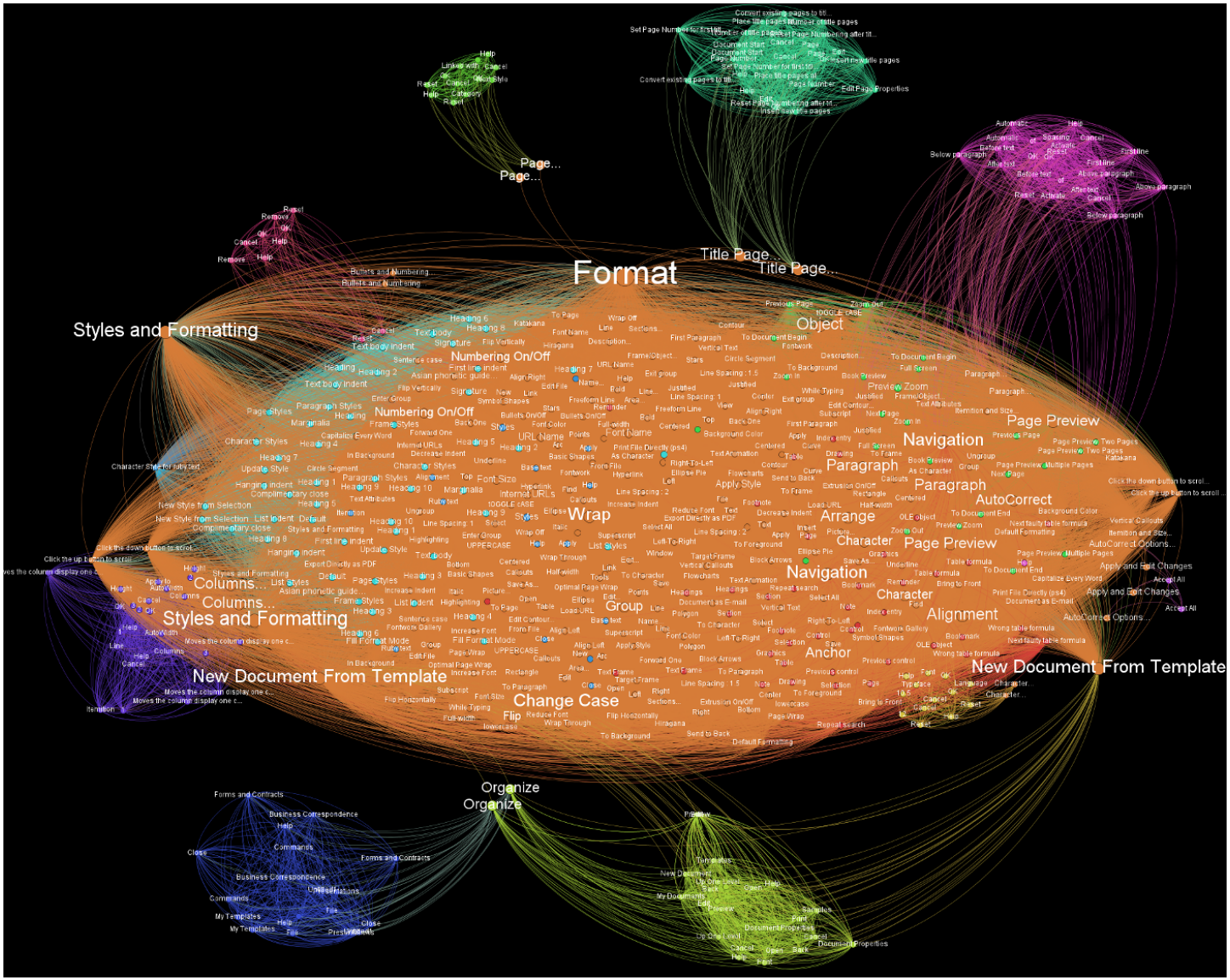


Extracting (Ripping) the Model

EFG for Part of ArgoUML



EFG for OpenOffice Writer



First Refinement

ICSE 2013 Tutorial

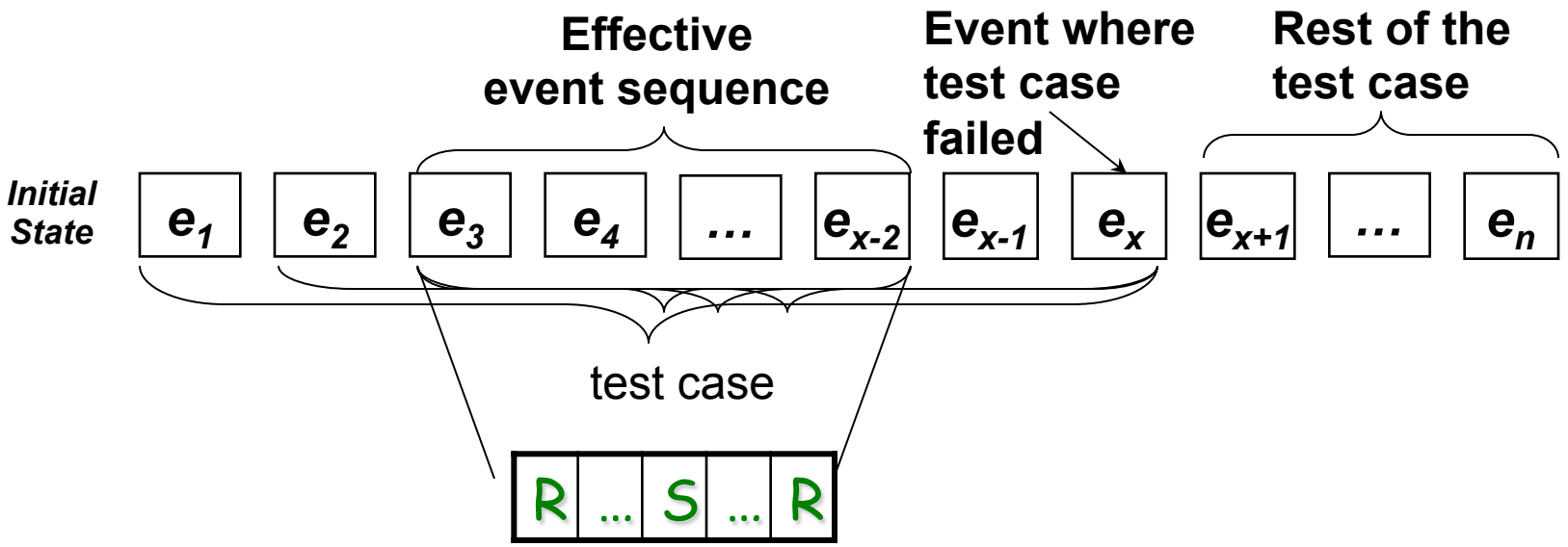
Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Event Interaction Graph
(EIG)

Failed Test Cases

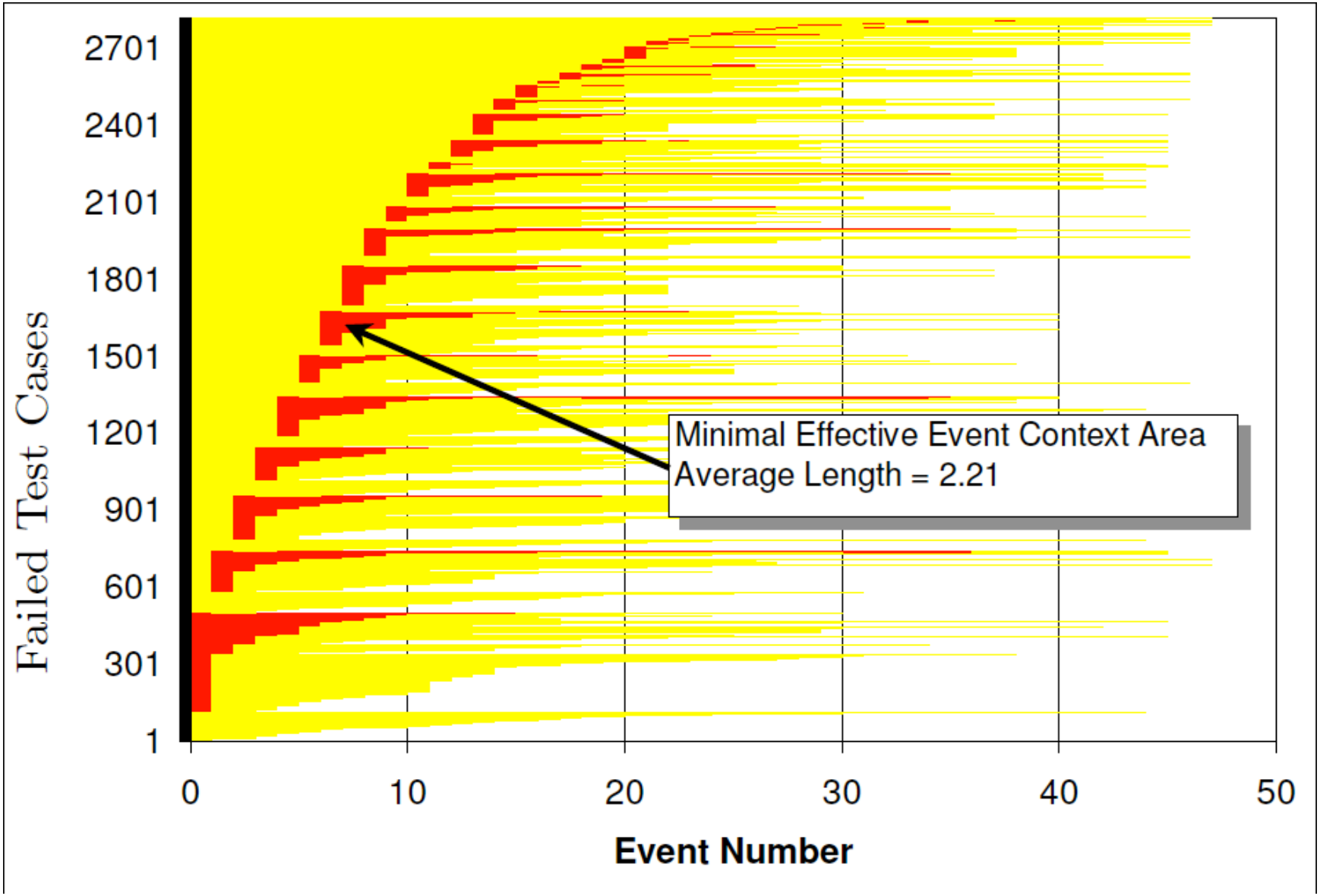
- Want to focus on test cases that reveal faults
- How to generate such test cases
- Remember we have Millions of test runs
  - Study the failed test cases
  - What caused them to fail?
  - Any common characteristics, structure
- Reduce the event-flow graph
  - represent “important” interactions

Dissecting Failed Test Cases



R = reaching events that open menus/windows
W = events that open windows
T= termination events that close windows
S = system-interaction events (e.g., CUT, COPY, PASTE)

Visualization from a Case Study



Understanding the Effective Event Sequence

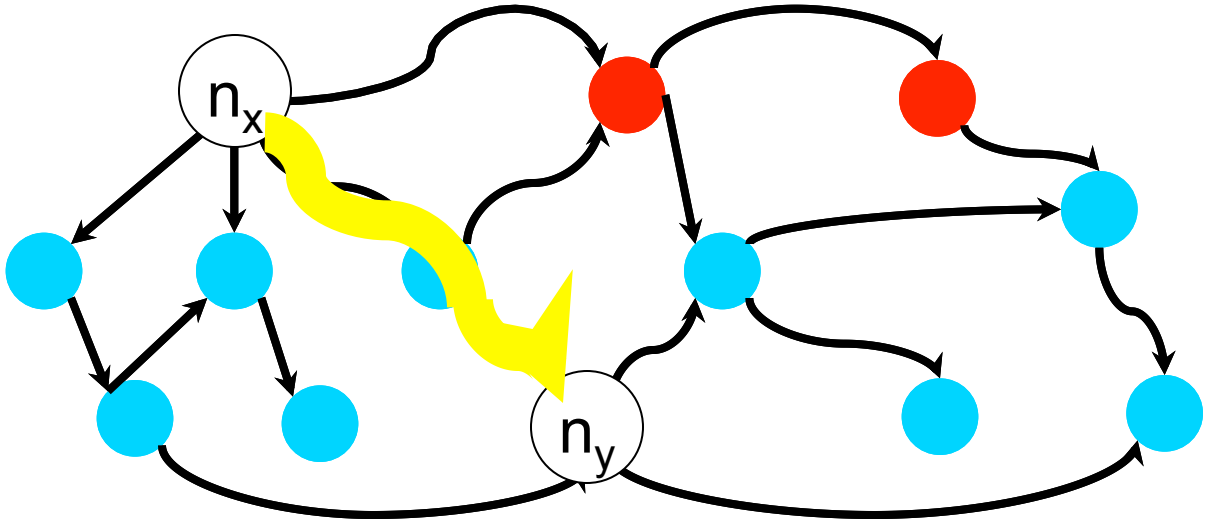
Pattern	Effective Event Sequence Structure	e_x	# Failures
1	R*	S	676
		W	6
2	R*S	S	431
		W	1
3	R*SR+	S	19
4	R*SR*(SR*)+	S	142

R = reaching events that open menus/windows
W = events that open windows
T= termination events that close windows
S = system-interaction events (e.g., CUT, COPY, PASTE)

Generate these effective sequences automatically

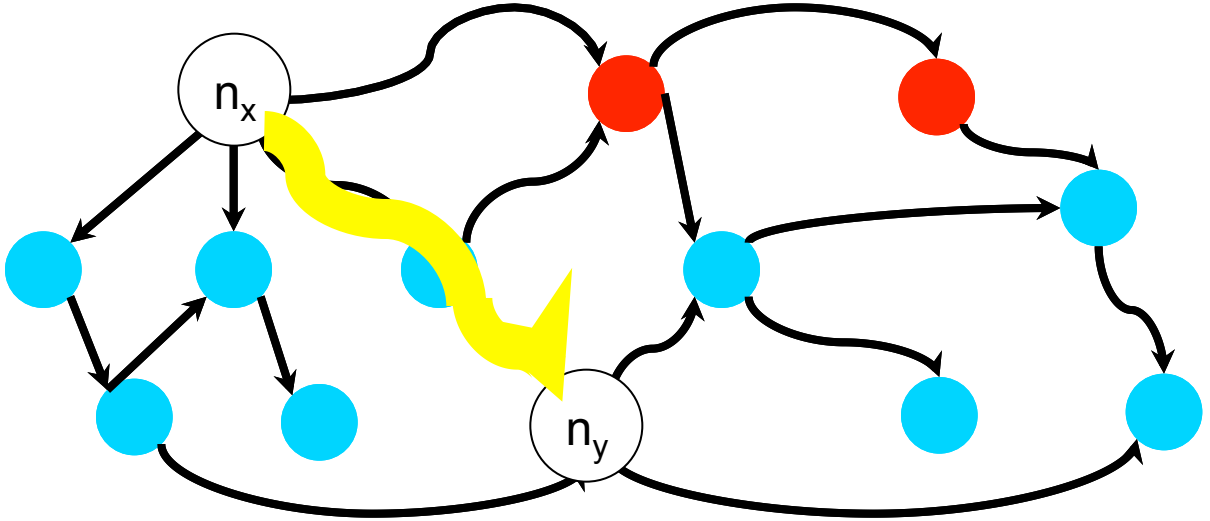
Definitions

Definition: There is an *event-flow-path* from node n_x to node n_y iff there exists a (possibly empty) sequence of nodes $n_j; n_{j+1}; n_{j+2}; \ldots; n_{j+k}$ in the event-flow graph E such that $\{(n_x, n_j), (n_{j+k}, n_y)\} \subseteq edges(E)$ and $\{(n_{j+i}, n_{j+i+1}) for\ 0 \leq i \leq (k-1)\} \subseteq edges(E)$. □



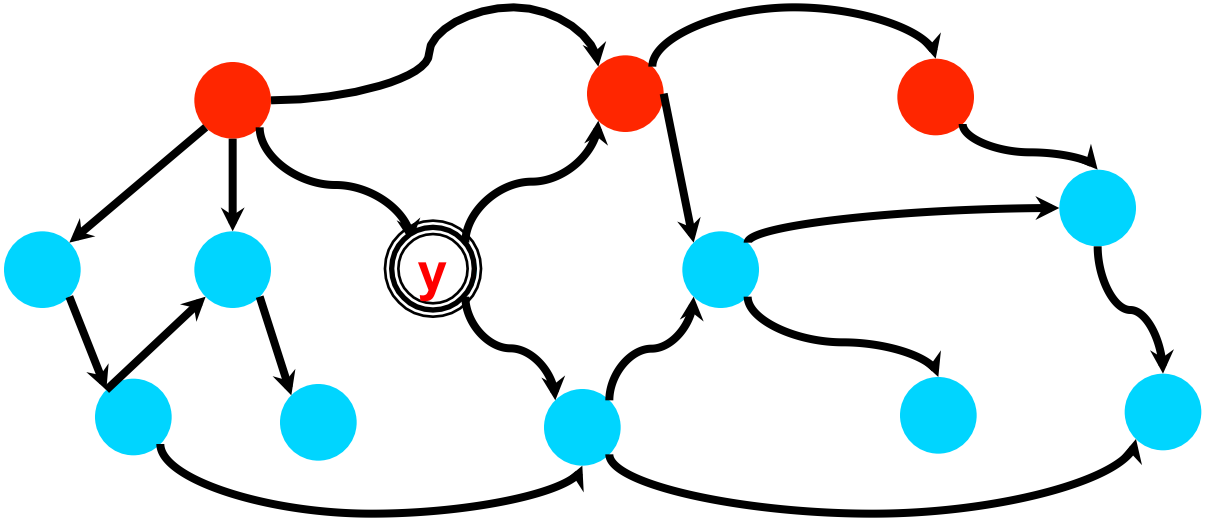
Definitions

Definition: An event-flow-path $< n_1; n_2; \ldots; n_k >$ is *interaction-free* iff none of $n_2, \ldots, n_{k-1}$ represent termination or system-interaction events. □



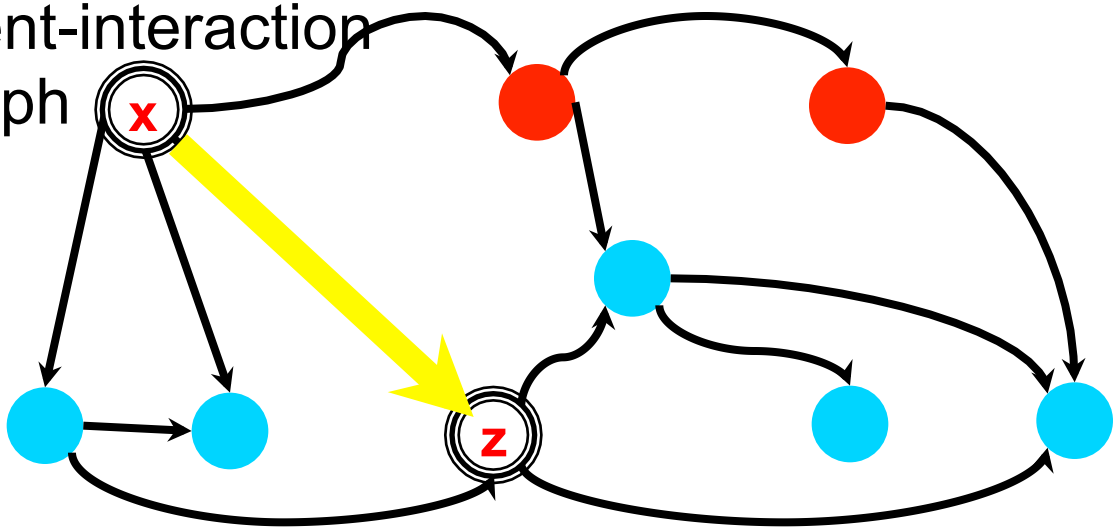
Definition: A system-interaction event (and termination event) e_x *interacts-with* system-interaction and termination event e_y iff there is at least one interaction-free event-flow-path from the node n_x (that represents e_x) to the node n_y (that represents e_y). □

EFG to EIG &Test Case Generation



Event-flow Graph

Event-interaction Graph



Pattern 1: R*
Pattern 2: R*S
Pattern 3: R*SR+
Pattern 4: R*SR*(SR*)+

Advantages of EIG

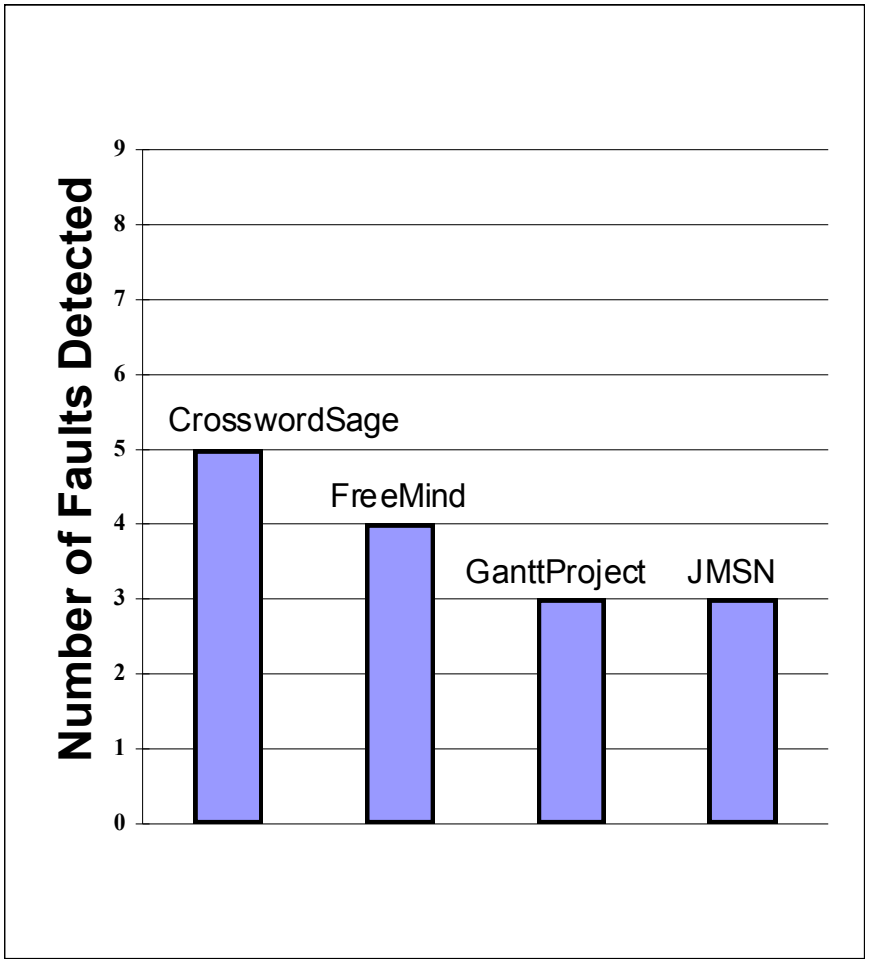
- Higher level of abstraction than event-flow graphs
  - Edges represent longer “important” paths in the GUI
- New test adequacy criteria
  - Event-flow graph interaction-free path coverage
  - Event-interaction graph edge coverage
- Obtain fully automatically from EFG

Full Automation

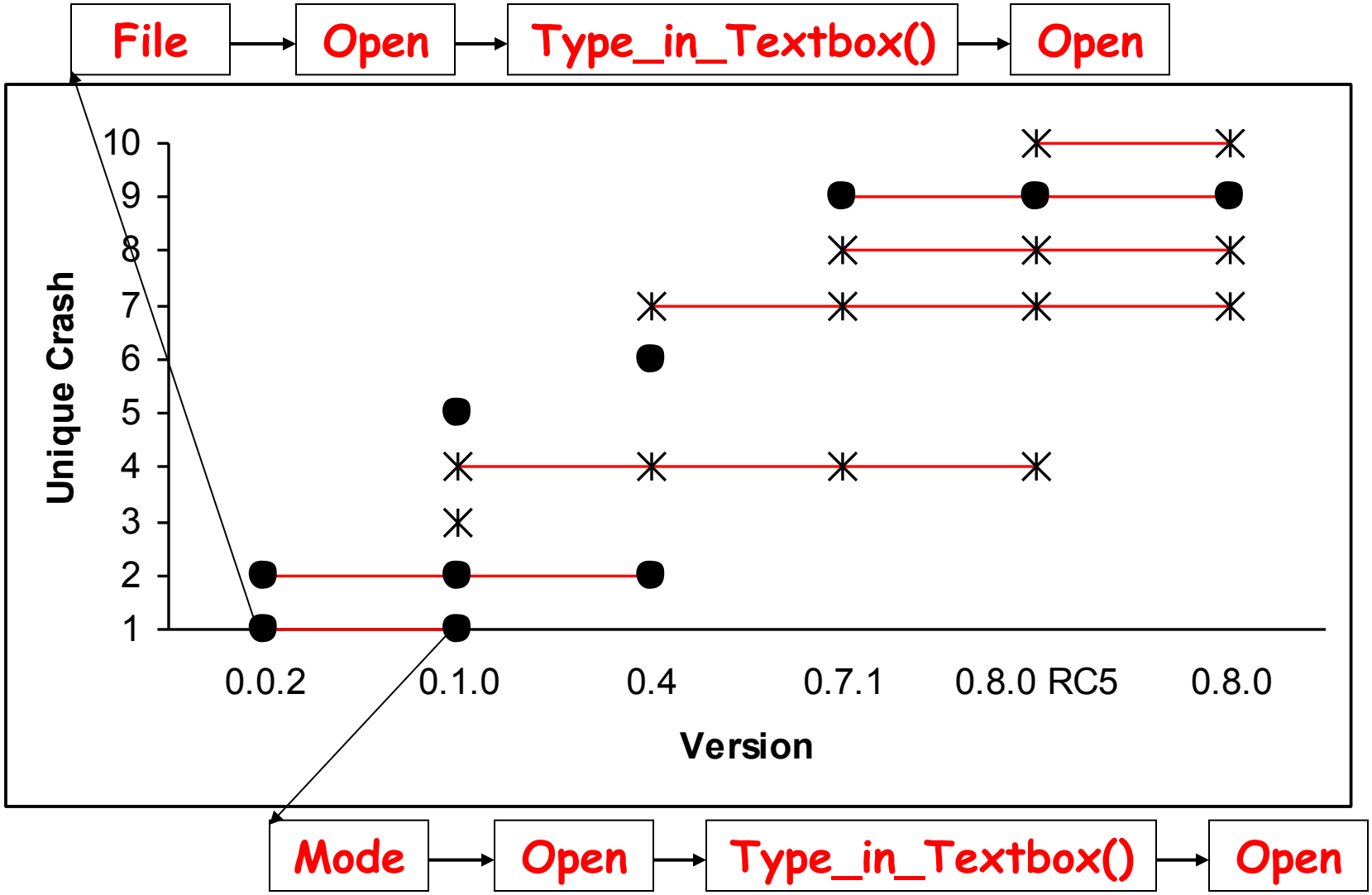
- Process
  - Reverse engineer application
  - Generate event-flow graph
  - Transform to event-interaction graph
  - Use our new test-adequacy criteria to generate test cases (e.g., cover all edges – important sequences of events in a GUI)
  - Use test executor to run all test cases
- Test Oracle
  - Assertions in the code
  - Invariants - Diakon
  - “Did the application crash?”

Lets See How It Works!

- Point to the CVS head
  - Push the button
  - Read error report
- What happens
  - Gets code from CVS head
  - Builds
  - Reverse engineers the event-flow graph
  - Creates EIG
  - Generates test cases to cover all the edges
    - 2-way covering
  - Runs them
- SourceForge.net
  - Four applications



Multiple Versions of FreeMind



Second Refinement

ICSE 2013 Tutorial

Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Event Semantic Interaction Graph

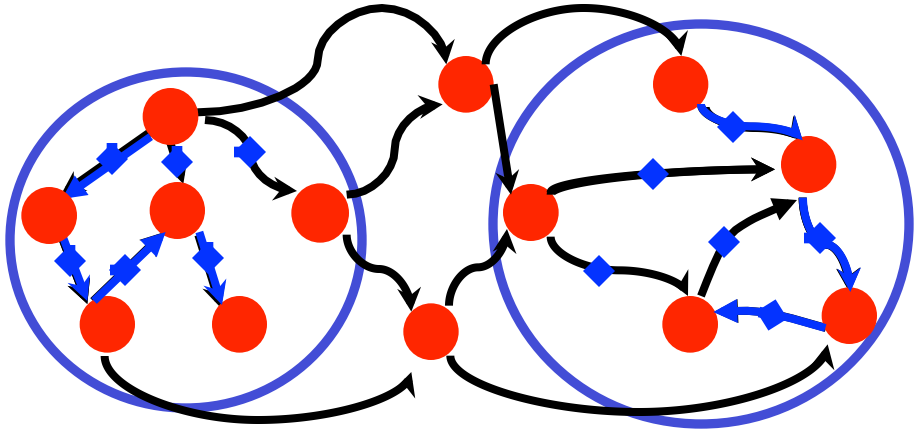
ESIG

Problems with EIG

- Although smaller than EFG, an EIG is still very large
  - Impossible to generate/execute long test cases systematically
- Need better ways to further sample event space

Digging Deeper!

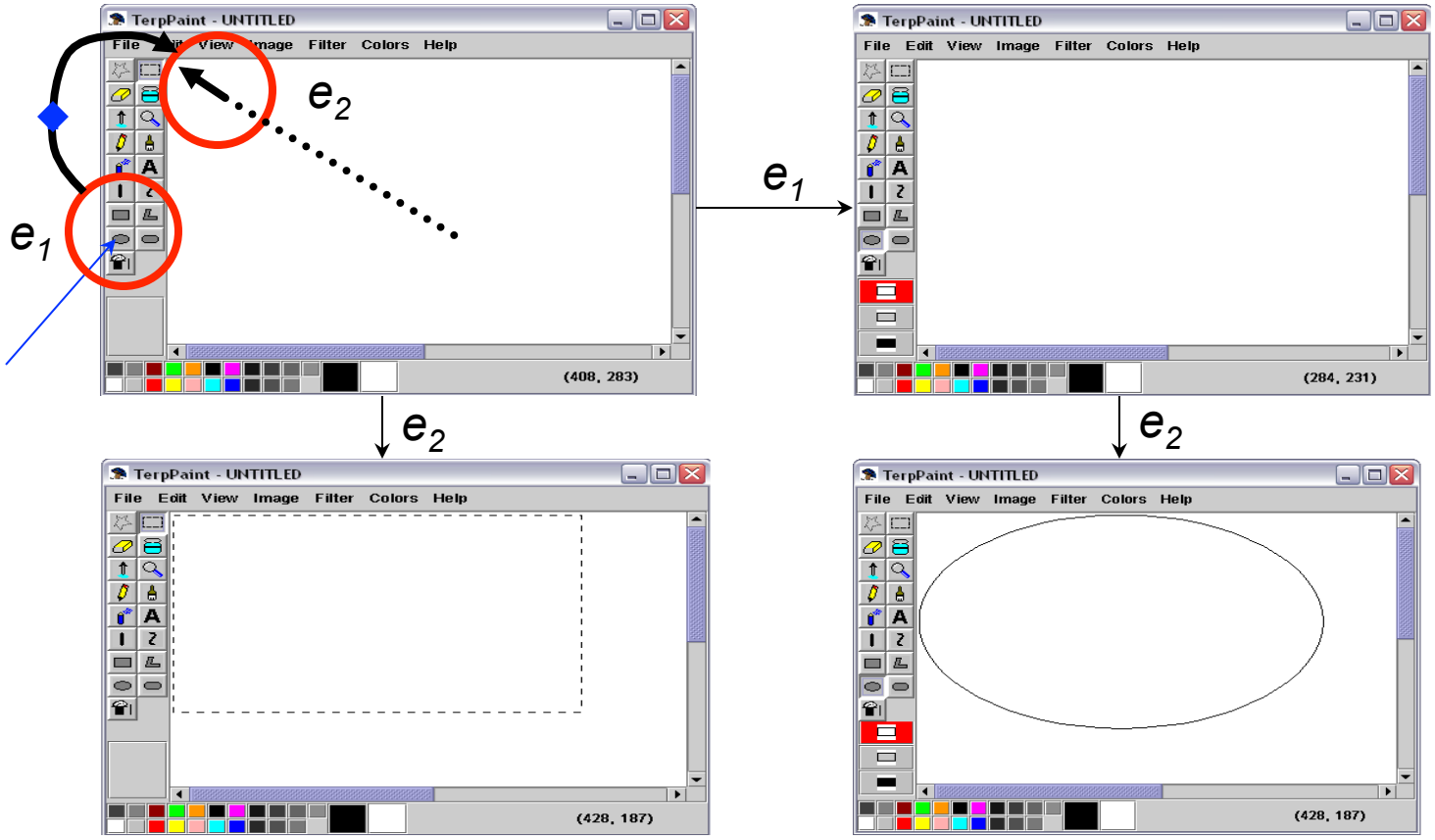
- **Intuition**
  - Non-interacting events (e.g., Save, Find)
  - Interacting events (e.g., Copy, Paste)
- **Key Idea**
  - Identify interacting events
  - Mark the EIG edges (Annotated graph)
  - Generate 3-way, 4-way, … covering test cases for interacting events only



EIG

Identifying Interacting Events

- High-level overview of approach
  - Observe how events execute on the GUI
  - Events interact if they influence one another's execution
    - Execute event e2; execute event sequence <e1, e2>
    - Did e1 influence e2's execution?
    - If YES, then they must be tested further; annotate the <e1, e2> edge in graph
- Automatic Process
  - Generate seed suite
    - 2-way covering test cases from EIG
  - Run test cases
    - Needed to obtain sets of GUI states
  - Collect GUI run-time states as feedback
  - Analyze feedback and obtain interacting event sets
  - Generate new test cases
    - E.g., 3-way, 4-way, … covering test cases



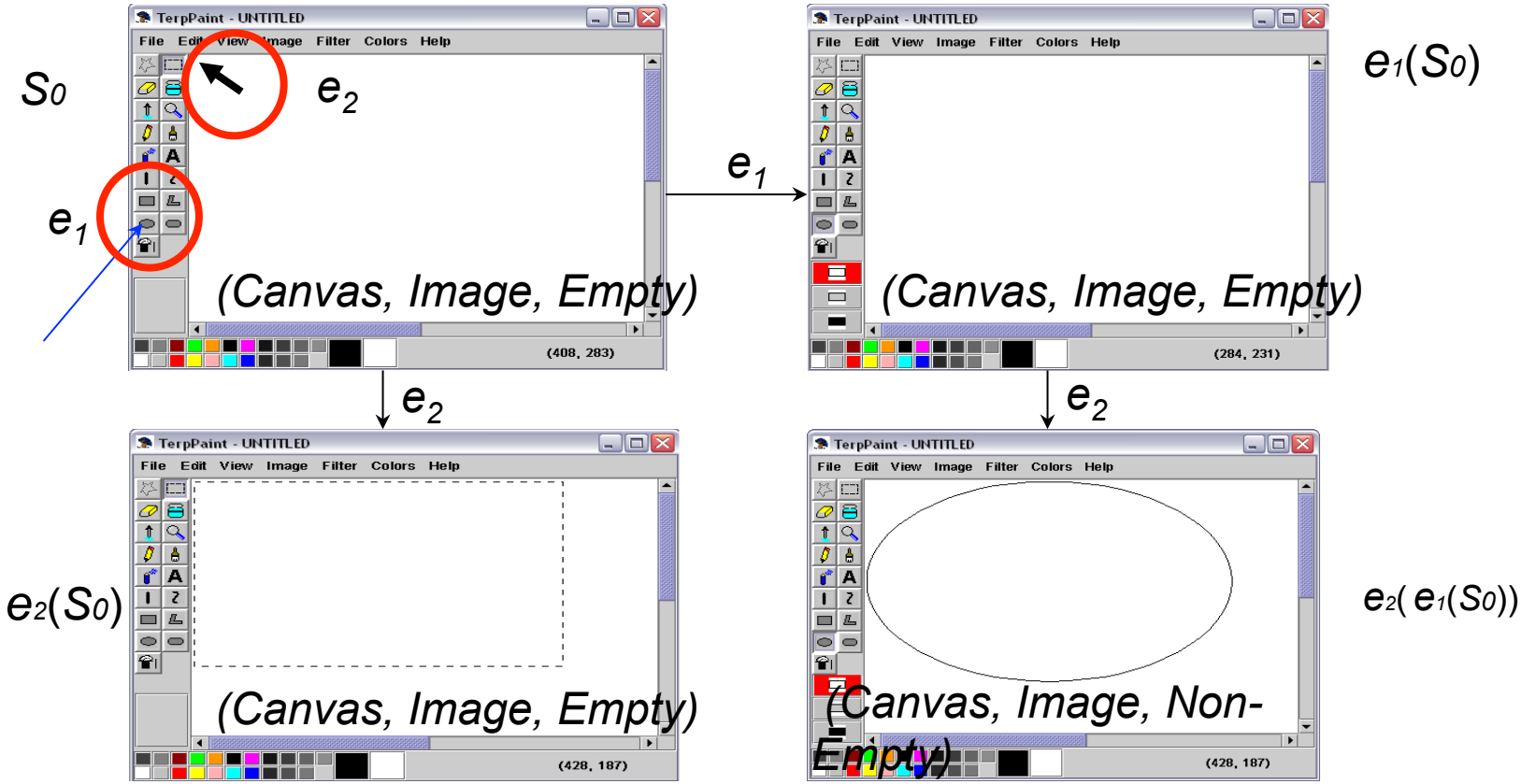
e_1:: select ellipse tool

```
public void ellipsePerformed (java.awt.event.ActionEvent evt){
...;    currentTool = toolEllipse; ... }
```

e_2:: drag mouse on canvas

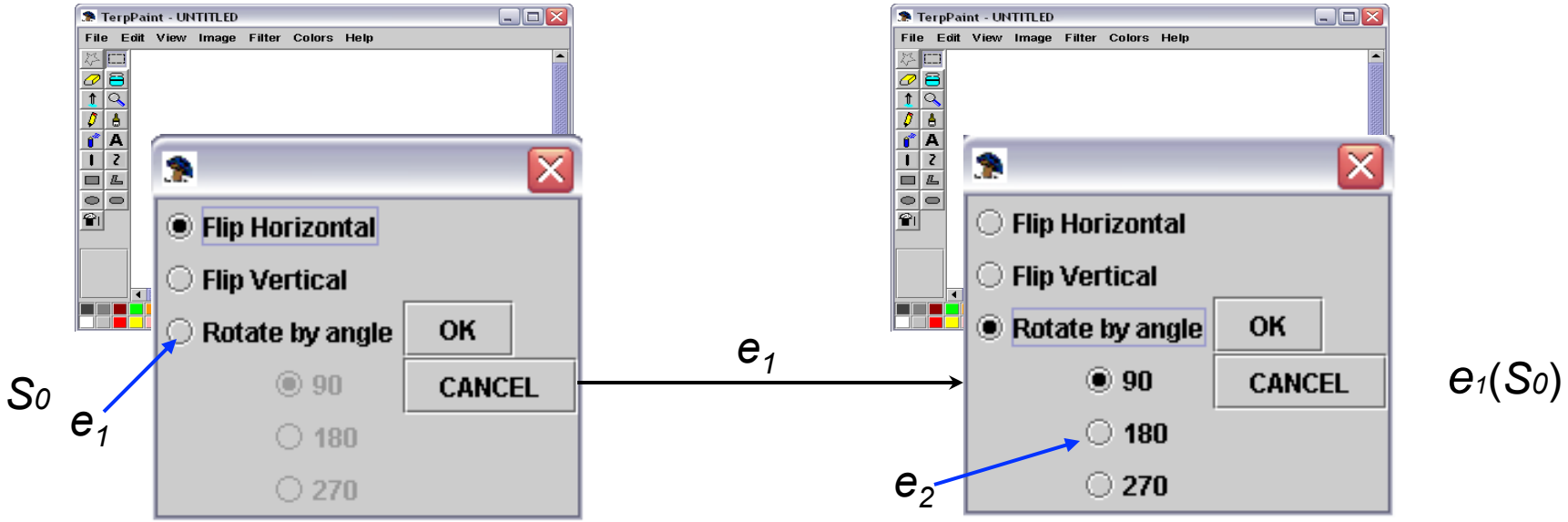
```
public void mouseDragged(java.awt.event.MouseEvent evt) {
...;    currentTool.dragAction(newEvt, center); ... }
```

Formalize Event Interactions



Predicate 1: $\exists w \in W; p \in P_W; v \in V_P; v' \in V_P;$ s.t: $((v \neq v') \wedge ((w, p, v) \in \{S_0 \cap e_1(S_0) \cap e_2(S_0)\}) \wedge ((w, p, v') \in e_2(e_1(S_0))))$;

Formalization (Cont.)



Predicate 12: $\exists w \in W$; *ENABLED* $\in P_W$;
TRUE $\in V_P$; *FALSE* $\in V_P$;
s.t: $(((w, ENABLED, FALSE) \in S_0) \wedge$
$((w, ENABLED, TRUE) \in e_1(S_0)) \wedge$
$EXEC(e_2, w))$;

More Cases

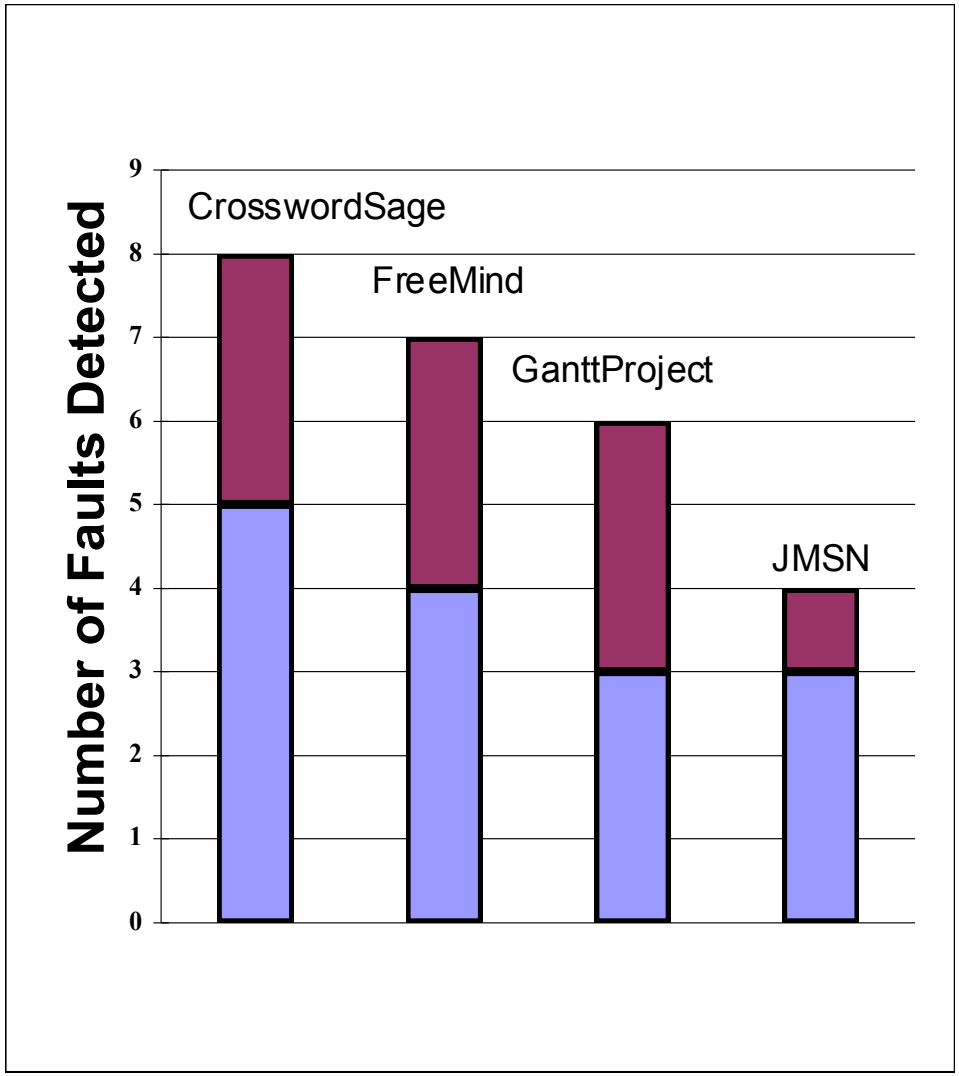
e_1's execution ***influences*** e_2's execution

For example, when e_2 is executed with e_1:

- e_2 modifies different widgets
- e_2 modifies the same widgets differently
- e_2 creates new widgets
- e_2 removes existing widgets or e_1 created widgets
- e_2 recreates widgets removed by e_1
- ...

Did We Do Better?

- Implemented new approach
- Fully automatic
  - Generate and execute all length-2 sequences
  - Analyze run-time feedback
  - Annotate EIG (called ESIG)
  - Use ESIG to generate longer test cases
- Compared feedback-based approach to EIG



Third Refinement

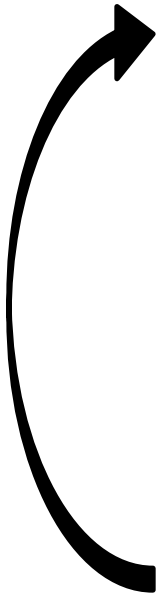
ICSE 2013 Tutorial

Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Alternate test generation and execution

Alternate Generation and Execution

- ALT is invented
  - Generate seed suite
  - 2-way covering test cases from EIG
  - Run test cases
  - Needed to obtain sets of GUI states
  - Collect GUI run-time states as feedback
  - Analyze feedback and obtain interacting event sets
  - Generate new test cases
  - E.g., 3-way, 4-way, … covering test cases



ALT Fault-Detection Effectiveness

Subject Application	Technique	i-way test suite		
		3	4	5
FreeMind	ESIG	☑☑	-	-
	ALT	☑☑	-	-
GanttProject	ESIG	☑☑☑☐	-	☐
	ALT	☑☑☑☑	-	☑
jEdit	ESIG	☑☑	☐	-
	ALT	☑☑	☑	-
OmegaT	ESIG	-	-	-
	ALT	-	-	-

Summary

- In situations where
  - It is impractical to create an accurate model by hand
    - E.g., Space is too large
  - Or the model is not known
- Start with an approximate model
  - Use reverse engineering
  - Heuristics
- Refine iteratively as well as test
  - Improve test cases with each iteration
  - Refine the model
    - Use for other testing activities
      - E.g., regression testing

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