AUTOMATED MODEL-BASED TESTING OF WEB APPLICATIONS

Oluwaseun Akinmade & Prof. Atif M. Memon
sheyakin@gmail.com atif@cs.umd.edu
Department of Computer Science &
Institute for Advanced Computer Studies (UMIACS)
University of Maryland, College Park, Maryland, USA
Event-driven Software (EDS) in Action!

Events

Event Handlers

Registered Event Handlers

Can handle an event of type $e_1$

- EDS testing is challenging; test cases are sequences of events
- Event-interaction space to be tested is huge
  - Number of test cases grows exponentially with length (# of events)
  - We don’t yet know how to sample the space of all event interactions
Web Applications are Event Driven

User Events

Web Application

PushButton btnG

PushButton btnI
Event Handlers Across Tiers

Presentation Tier

Logic Tier

Data Tier

User Events

E1

E2

E3

E4

E5

E6

Data management events

• Many complex tests for data management tier
• Functional and performance tests
• Database state plays important role
• Requires experience and skill

Data Tier

May consist of many interrelated technologies; inter-tier events:
• CGI Applications
• ISAPI Extensions and Filters
• Active Server Pages
• ASP Applications
• Process Isolation and Crash Recovery

• All these tests (and more) are important.
• E.g., test for performance, concurrent users, environment (e.g., client configuration)

© Akinmade & Memon; @cs.umd.edu
Our Research Focus – Test Through the UI

- Obtain sequences of user events
  - These sequences become test cases
- Obtain expected outcome (for “test oracle”)
- Execute the test cases automatically
- Verify execution behavior

User Events

Web Application

PushButton btnG

PushButton btnI

© Akinmade & Memon; @cs.umd.edu
Environment-Influenced Accessibility

Mozilla 0.9

FEDERATION OF AMERICAN SCIENTISTS

Search Results:

© Akinmade & Memon; @cs.umd.edu
State of the Practice – Manual

- Very few test cases
- **Test oracle**: mostly visual
- Test “common” sequences
  - What is “common”?  
- Try some “uncommon” sequences

- Test cases not reusable
  - Must do it again when app changes
State of the Practice – Code Tests

WebDriver
- Can be replayed automatically
- Multiple machines
- Regression testing: Application evolves
- Oracle: what to check?
- Still have “Few tests” problem

Several other tools
- Selenium

```java
// Use Google.
driver.get("http://www.google.com");

// Search for Scott McMaster!
WebElement query = driver.selectElement("//input[@name='q']");
query.setValue("Scott McMaster");

// Click the search button.
WebElement submitButton = driver.selectElement("//input[@name='btnG']");
submitButton.click();

// Grab the results.
List<WebElement> results = driver.selectElements("//h2[@class='r']/a");
```

Thanks Scott!
State of the Practice – Capture/Replay

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

- Good example: Selenium IDE
  - Firefox addon that records clicks, typing, and other actions to make a test, which you can play back in the browser.
  - [http://selenium.openqa.org](http://selenium.openqa.org)

Set of Tasks

Capture

Tester File

Replay

Tester File

Tester File

Tester File
State of the Practice – Capture/Replay

- Tool replays tester’s actions
- Verifys application response against stored response
State of the World

• Solved Problems
  – Tests can be replayed automatically
  – Multiple servers
  – Run tests in multiple browsers and platforms
  – Good test management
  – Can vary data for a test case
    • Text fields

• These problems persist
  – How to sample the space of all possible event interactions
  – Regression testing: Web application evolves
  – Oracle: what to check?
  – More philosophical question remains
    • How to test Web software and other large event-driven applications?
Our Approach

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

- For Desktop GUI Applications
  - Probabilistic state machines
  - Preconditions & Effects (from specs)

- Test oracle
  - AI planning for test case generation
  - Directed graph models (user’s view)
  - Event-flow graphs
  - Dynamic behavior (from run-time)

- For Web Applications
  - State-machine Models (from specs)

**Web Application Model**

**Operator**: \textit{ZoomIn}

**Parameters**: \(wX\): window; \(m\): map;

**Precondition**:

\[\text{isCurrent}(wX) \& \text{currentZoom} = \text{getZoom}(m) \& \text{currentZoom} \neq \text{maxZoom}.\]

**Effects**:

\[
\text{currentZoom}++
\]
Our future plans for model-based web application testing...
A sneak preview!

BASED ON OUR PAST WORK ON
DESKTOP GUI APPLICATIONS
Modeling the User’s Event-Interaction Space

- **Event Flow graph (EFG)**
- Nodes: GUI events; Edges: *Follows* relationship

Very promising technology for GUI-like Web UIs
• How to create the event-flow graph?
  • Manually?
    • Too large for non-trivial GUIs
A Part of MS WordPad
Its Event-Flow Graph
• How to create the event-flow graph?
  • Automatically
    • GOAL: Obtain the Event-Flow Graph
    • Fully automatically
    • No source code
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
  - Java Swing API
  - Interaction between Java and the OS
- **Result** – Generic process for GUI Ripping
- **MS Windows, Java Swing**
- **Immediate impact** – Obtained EFGs for large GUIs in a few minutes
- **Automatic transformation to event-interaction graph (EIG)**

---

**Currently working on a Web Ripper**

If anyone here has any ideas, please talk to us!
Full Automation

• Process
  – Rip GUI 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
  – “Did the application crash?”
Let's See How It Works!

- **SourceForge.net**
  - Four applications

- **The Process**
  - 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

![Bar chart showing the number of faults detected for CrosswordSage, FreeMind, GanttProject, and JMSN.](chart.png)
Going Beyond 2-way Covering Tests

• Xun Yuan (PhD student, now at Google – and in the audience today)
• **Intuition**
  – Non-interacting events (e.g., Save, Find)
  – Interacting events (e.g., Copy, Paste)
• **Approach**
  – Identify interacting events
  – Mark the
    EIG edges
    (Annotated graph)
  – Generate
    3-way, 4-way, … covering
    test cases for interacting
    events only

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?
    • Use GUI runtime state to answer this question
    • If YES, then they must be tested further; annotate the <e1, e2> edge in graph

• Use feedback
  – Generate seed suite
    • 2-way covering test cases
  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

Did We Do Better?

- Compare feedback-based approach to 2-way coverage

![Bar chart comparing different tools]

- CrosswordSage: 9 faults detected
- FreeMind: 7 faults detected
- GanttProject: 6 faults detected
- JMSN: 4 faults detected
BACK TO WEB APPLICATIONS
Let's Start with an Example

- This simple web app has states
- Can be determined by executing the app
- Or examining requirements
- Or specs (if they exist)
- For example, the login page offers at least 4 user events
State-Machine Model

- Model traversal to obtain event sequences
- And intermediate states
- For example, start with S_index state
  - usernameText, <usrnameSt>
  - PasswdText, <usr+passSt>
  - Submit, <main.html>

- Many other ways to traverse
- These ways lead to test planning, determine cost, and test adequacy criteria
Example Event Sequences

elem.sendKeys(“username”);
elem.sendKeys(“password”);
elem.submit();

elem.sendKeys(“password”);
elem.sendKeys(“username”);
elem.submit();

elem.sendKeys(“Hello World!”);
elem.toggle();

// t_agree

// t_browse

elem.click();
elem.clear();

// del_browse

elem.sendKeys(“Hello World!”);

elem.clear();

// del_text

elem.click();

elem.clear();

// del_text

elem.clear();

// del_browse

elem.clear();

// del_text

elem.toggle();

// del_agree

elem.toggle();

// t_agree

elem.submit();

elem.click();

// t_prev

elem.click();

// t_home_link

elem.click();

// t_browse

elem.sendKeys(“Hello World!”);

elem.submit();
Test Cases = Event Sequences + Expected State

// Event/User Action
passwd_elem.sendKeys("password");
// Expected state is passwdSt
assert (
    (driver.getTitle().equals("Index Page")) &&
    (username_elem.getValue().equals("")) &&
    (passwd_elem.getValue().equals("password")))

// Event/User Action
reset_elem.click(); // t_reset
// Expected state is S_index
assert (
    (driver.getTitle().equals("Index Page")) &&
    (username_elem.getValue().equals("")) &&
    (passwd_elem.getValue().equals("")))

// Event/User Action
passwd_elem.sendKeys("username");
// Expected state is usernameSt
assert (
    (driver.getTitle().equals("Index Page")) &&
    (username_elem.getValue().equals("username")) &&
    (passwd_elem.getValue().equals("")))

Example adequacy criteria: cover all states; cover all transitions; all possible pairs of states
From [http://research.microsoft.com/fse/asml/](http://research.microsoft.com/fse/asml/)

AsmL is the Abstract State Machine Language. It is an executable specification language based on the theory of Abstract State Machines, invented by Yuri Gurevich. AsmL provides the foundations of the model-based testing tool Spec Explorer.
Our Tool – STATEST (2)

- ASML Code
- Our Automatic Event-Sequence Generator
- Event Sequences + State
- Our WebDriver Command Generator
- WebDriver Test Cases

1. State-Code Mapping
2. Event-Code Mapping
Event-Sequence Generation Algorithm

1. Begin at the Start state
2. Call getShortestExample() method to get a traversal_path
3. To recursively get a list of all other possible shortest paths:
   1. Make a copy (tempModel) of the currModel Automaton
   2. For each Transition pTran in the current traversal_path
      a. Remove pTran from the tempModel Automaton
      b. Remove all dead states as a result of the above action
      c. Call getShortestExample() method to get a new traversal_path
      d. Make a new copy (tempModel) of the Automaton
4. Recursively traverse the state-machine model to get a list of longer path sequences that could possibly be generated from the currModel Automaton. This is modeled after the Depth First Search algorithm.

dfs_Func(state, pathSeq):
   1. If state is an accepting/final state { add pathSeq to pathList array }
   2. Otherwise:
      a. If all outgoing transitions from state have been taken
         i. Backtrack to previous state node
      b. Else for each outgoing transition trans from state that is not taken
         i. Add trans to pathSeq
         ii. next_state = trans.destState()
         iii. dfs_Func (next_state, pathSeq)
1. To make sure that there are no states/transitions omitted from the pathList generated above:
   1. Get list of all transitions (allTrans) in the currModel Automaton
   2. For each transition trans in allTrans that is not in pathList: Add trans to missedTrans list
   2. For each transition mTrans in missedTrans: generate path sequences that include mTrans.
<S_index>, PasswdText, <passwdSt>
<passwdSt>, UsrnameText, <usr+passSt>
<usr+passSt>, Submit, <Main.html>
<S_index>, PasswdText, <passwdSt>
<passwdSt>, UsrnameText, <usr+passSt>
<usr+passSt>, Submit, <Main.html>
<usr+passSt>, Del_passwd, <usrnameSt>
<S_index>, PasswdText, <passwdSt>
<passwdSt>, UsrnameText, <usr+passSt>
<usr+passSt>, Submit, <Main.html>
<usr+passSt>, Del_Passwd, <usrnameSt>
<usrnameSt>, PasswdText, <usr+passSt>
\(<S\_index>, \text{PasswdText}, <\text{passwdSt}>\)

\(<\text{passwdSt}>, \text{UsrnameText}, <\text{usr+passSt}>\)

\(<\text{usr+passSt}>, \text{Submit}, <\text{Main.html}>\)

\(<\text{usr+passSt}>, \text{Del\_Passwd}, <\text{usrnameSt}>\)

\(<\text{usrnameSt}>, \text{PasswdText}, <\text{usr+passSt}>\)

\(<\text{usr+passSt}>, \text{Del\_Usrname}, <\text{passwdSt}>\)
<S_index>, PasswdText, <passwdSt>
<passwdSt>, UsrnameText, <usr+passSt>
<usr+passSt>, Submit, <Main.html>
<usr+passSt>, Del_Passwd, <usrnameSt>
<usrnameSt>, PasswdText, <usr+passSt>
<usr+passSt>, Del_Usrname, <passwdSt>
<passwdSt>, Reset, <S_index>
<S_index>, PasswdText, <passwdSt>
<passwdSt>, UsrnameText, <usr+passSt>
<usr+passSt>, Submit, <Main.html>
<usr+passSt>, Del_Passwd, <usrnameSt>
<usrnameSt>, PasswdText, <usr+passSt>
<usr+passSt>, Del_Usrname, <passwdSt>
<passwdSt>, Reset, <S_index>
<S_index>, Reset, <S_index>
© Akinmade & Memon; @cs.umd.edu

\begin{enumerate}
\item \textbf{S\_index}, PasswdText, \textbf{<passwdSt>}
\item \textbf{<passwdSt>}, UsrnameText, \textbf{<usr+passSt>}
\item \textbf{<usr+passSt>}, Submit, \textbf{<Main.html>}
\item \textbf{<usr+passSt>}, Del\_Passwd, \textbf{<usrnameSt>}
\item \textbf{<usrnameSt>}, PasswdText, \textbf{<usr+passSt>}
\item \textbf{<usr+passSt>}, Del\_Usrname, \textbf{<passwdSt>}
\item \textbf{<passwdSt>}, Reset, \textbf{<S\_index>}
\item \textbf{<S\_index>}, Reset, \textbf{<S\_index>}
\end{enumerate}
TOOL DEMO
Helping the Tester Create the Model

Application Under Test

Our Web Ripper (state-machine based)

1. State-Code Mapping

2. Event-Code Mapping

3. State-Machine Tables

- State-Code Mapping
  - Color
  - Day
  - Time
  - Color + Day
  - Color + Time
  - Day + Time
  - Color + Day + Time

- Event-Code Mapping
  - Error
  - Main
  - Submit

© Akinmade & Memon; @cs.umd.edu
Creating the State Machine

• State discovery – Web Ripper
• Definition of State
• Steps
  – Automatic state discovery
  – Manual inspection, editing

• DEMO
What’s Next?

• Models, Algorithms, and Evaluation
  – Enhance/extend the model
  – New state-machine traversal strategies
  – How good are our new test cases?

• Limitations also point to future work
  – Plug-ins (e.g., Flash)
  – Nature of events (discrete vs. drag & drop)
  – Popups and unexpected widgets

• GUI Testing background points to lots more work 😊
Questions, Discussion and Food for Thought

How do we test this familiar web application?