Test Automation at the GUI level

Tanja E.J. Vos

What is GUI testing?

Testing at the GUI Level
- GUI is where all functionality comes together
- Interacts with the underlying code
- The whole system can be executed by means of the GUI
- Integration / System Testing

Testing at the GUI Level
- Most applications have GUIs
  - Computers, tablets, smartphones....
  - Even safety critical applications

What is a GUI?
- Contains graphical objects, called widgets
- Menus, textboxes, buttons, scrollbars

Faults that arise at UI level are important
- These are what your client finds
- GUI tests from their perspective!
Widgets form a hierarchy the widget-tree.

Typical Widget Tree

Widgets have properties which have values at run-time.
GUI state
- The widget tree
- The values of the properties of each widget

GUI action
- Users can exercise actions (click, type, drag, drop,...)
- These cause a state change

What is GUI testing
- Specify test sequences
  - Sequences of GUI actions
  - Click, drag, drop, type
  - Provide input where needed (e.g., filling text fields)
- Specify oracle
  - The test oracle
  - The correct states after execution of each action

Specify test sequences
Specify oracle
Together they test a requirement

Step 1
Open MS Word

Step 2
Click on menu View

Step 3
Click on Media Browser

Step 4
Select a picture and drag into the document

After each step:
- No failure has occurred
- No error message has popped up

After last step:
- The picture is in the document
Manual testing

- Tedious
  - Executing the same clicks over and over again
- Tiresome and boring
  - Repeating the same tests after changes in the UI
  - Filling the same form over and over again
- Regression testing
- Error prone
- Costly

State of practice: make scripts

- Test sequences
- Oracles
- Develop scripts
- Execute
- Maintenance

Capture & Replay

- Tools: Captures user interaction with the UI
- Records a script
- That can be automatically replayed

Examples

- Open source
  - Selenium
  - Abbot
- Commercial
  - QF-Test
  - Rational Functional/Robot Tester (IBM)
Capture & Replay

- Advantages
  - Simple and easy

- Disadvantages
  - Scripts break as GUI changes
  - Maintenance problem

- These are huge problems
  - GUIs change all the time
  - Requirements tool

Visual testing (VGT)

- Based on image recognition

- Advantages
  - Easy to understand
  - Hardly no programming skills needed
  - Solves part of maintenance problem
    - Robust against some changes
    - But not all
      - Move Media Browser within same menu: YES
      - Move Media Browser to another menu: NO
      - Change the icon: NO
  - Studies show maintenance still an issue

Visual testing

- Easy to understand
- Hardly no programming skills needed
- Solves part of maintenance problem
  - Robust against some changes
  - But not all
    - Move Media Browser within same menu: YES
    - Move Media Browser to another menu: NO
    - Change the icon: NO
  - Studies show maintenance still an issue

What is Test*?

Our contribution: Test*

- Scriptless
  - What is not there does not need to be maintained

- Departs from random testing
  - Immediately start testing without requirements

How?
We can start automated testing

- Immediately (minimal set-up)
- No scripts
- No maintenance here
  - The widget tree is extracted in each new state
  - If the state is different, so is the widget tree

100% Automated online oracles

- Crashes
- Hangs

Out of the box

Robustness testing

- Online oracles for suspicious titles and outputs
- Specify them with a regular expression

Oracle – Suspicious titles
(under the hood)
ClaveCon

- Spanish SME
- ERP system
- Written in Visual Basic
- Microsoft SQL Server 2008 database
- Targets the Windows operating system

<table>
<thead>
<tr>
<th>TESTAR</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td>26 hour</td>
</tr>
<tr>
<td>Testing</td>
<td>91 hour</td>
</tr>
<tr>
<td>Post testing</td>
<td>1.5 hour</td>
</tr>
<tr>
<td>Critical faults</td>
<td>10</td>
</tr>
</tbody>
</table>

SOFTEAM

- French and large company
- Backend system for virtualization
- 80% SoVW re-object existing faults

\[
FDR = \frac{\text{cum effective/fault}}{\text{num of injected faults}} \times 100\%
\]

<table>
<thead>
<tr>
<th>TESTAR</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td>40 hour</td>
</tr>
<tr>
<td>Testing</td>
<td>77 hour</td>
</tr>
<tr>
<td>Post testing</td>
<td>3.5 hour</td>
</tr>
<tr>
<td>FDR</td>
<td>61%</td>
</tr>
<tr>
<td>Code coverage</td>
<td>70%</td>
</tr>
</tbody>
</table>

Cap Gemini/
ProRail

- Dutch cooperation
- Web GUI
- System for managing the assignment of train platforms

<table>
<thead>
<tr>
<th>TESTAR</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preparation</td>
<td>44 hour</td>
</tr>
<tr>
<td>Testing</td>
<td>51 hour</td>
</tr>
<tr>
<td>Post testing</td>
<td>5 hour</td>
</tr>
<tr>
<td>Critical faults</td>
<td>4</td>
</tr>
<tr>
<td>Functional</td>
<td>80%</td>
</tr>
<tr>
<td>coverage</td>
<td>73%</td>
</tr>
</tbody>
</table>

Beside these Test*

- Microsoft office suite
- Bitrix 24
- Test the test tool TESTONA (eclipse based)
- Over 10 web applications of Spanish companies
- 12 students currently working on it
- Several companies doing proof of concepts

How does it change testing?
How?

Application/Domain specific oracles

Need to be programmed/specified

We cannot avoid making oracles manually

TESTAR shares this problem with ALL automated approaches

Oracle problem

How does it change testing?

How is the test effort distributed

How can Test* change testing?

Random testing

“Valuable test case generation scheme”


“Necessary final step in the testing activities”


“Probably the poorest testing method”

Use partition testing

**In the 70s**

Use domain knowledge of the SUT to partition
Group together similar test cases
Choose one

**Random testing**

**In the 80s**

Düchen and Wohlin (1994):
Simulation and experiments showing random better
than systematic partition testing.

Hamlet and Taylor (1988):
More experiments showing the same
Counterintuitive.

**In the 80s**

Random testing

**Counterintuitive**

- Why do random testing and systematic testing seem to be almost on par?
- What are the properties of random testing?
- When is random testing more effective than partitioning and the other way around?

**In the 80s**

Random testing

- Weyuker and Jeng (1991)
  - Partition well or better not partition at all!
  - $D_i = \text{probability that a test randomly drawn from } D$ fails
  - if $D_1 = D_2 = D_3 = \ldots = D_k$ then $P_r = P_p$
  - In other cases it can be better or worse,....

**From the 90s on**

- Chen and Yu (1996)
  - If the number of test cases from $D_i$ proportional to its cardinality
  - then partition testing cannot perform worse than random
- Gutjahr (1999)
  - Failure $D_i$ rates are probabilistic and not deterministic.
  - We base our testing strategy only on expectation $E$ not actual cases
  - They show:
    - if $\sigma_1 = \sigma_2 = \sigma_3 = \ldots = \sigma_k$ then $P_r \leq k P_p$
  - But, no further advice on how to make it better
Arcuri et al (2012)
Random testing as an instance of coupons collector problem!
And so results from that theory can be re-used!

Oatmeal boxes come with coupons with n different animals.
\( X = \) the amount of boxes of oatmeal do we need to buy to collect all n animals
Animals = test targets \{T_1, ..., T_n\}
Boxes = test cases
We get lower bounds for the amount of test cases needed:
\( E[X] = \Omega(n \log n) \)
\( p = \) the probability that a random test case covers \( T_i \)
Uniform \( p \), then even tighter bound:
\( E[X] = \Theta(n \log n) \)

Böhme and S. Paul (2016)
For automated GUI testing.....
Generating test case is:
- Specification
- Capture (or automate with script)
- Maintenance!!
And random selection gave us quite good results on the software we tested......
Can we do better?

How can we find more faults?
Some test cases might be more likely to reveal faults
Don’t pick at random, but try to optimize criteria
What criteria?

Where can we find faults?
- Surrogate measures
- We cannot measure % of faults found
- We measure something we believe, hope or have shown to be correlated to that attribute.
- Coverage
- Diversity
- Novelty
Let the testing tool learn by itself how to test better!!
Surrogate measures
- as many different actions as possible?
- make large call trees?
- visit as many different states as possible?
- make long sequences?
- find novel states?
- We need to investigate many more

Machine Learning (Q-learning)
- sets $S$ of possible states
- sets $A$ of possible actions
- description $T$ of the effect of action in a state
  $T : S \times A \rightarrow S$
- state $s$ then select an action from $a \in A$ that causes a transition to a next state $s'$
- reward function $R : S \times A \rightarrow \mathbb{R}$

Find a policy $\pi$ which maximizes the reward by selecting an appropriate action in each state

Rewards
- Set $S$ of possible states the SUT can be in
- For all $s \in S$, we have sets $A_s \subseteq A$ of actions
- We focus is on exploration of the GUI
- We reward actions $a$ with low execution count $\text{ec}(a)$

$$\forall s \in S, a \in A_s: R(s,a) = \begin{cases} R_{\text{max}}, & \text{if} \ \text{ec}(a) = 0 \\ 0, & \text{otherwise} \end{cases}$$

Q-learning algorithm

Q-learning algorithm

Tried it out – ClaveiCon
Call trees

- The larger the call tree, the more aspects of the SUT are tested (McMaster and Memon)
- Bytecode instrumentation of the SUT to obtain the call tree, so no source code needed

Ant Colony Optimization

- Collectively ants can solve complex tasks
- Ants communicate using pheromones
  - They lay this on their path
  - Pheromone trail strength accumulates when multiple ants use a path
  - Other ants go where there is good pheromone strength

Ant Colony Optimization

- We have a population of ants
- Set of choices \( C \) (= actions)
- The ants generate trails (= test sequences)
- By choosing \( c \) according to pheromone values \( p \) (= selection criteria)
- Choices (= actions) that appear in "good" trails (= max call tree) accumulate pheromones

We do not want to make an instance for each action selection strategy!

The tools should learn what the best action selection strategy is!

Evolve action selection rules
**Action selection rules**

- IF-THEN
- IF-THEN-ELSE

**Mutation**

**Crossover**

**Evolve action selection rules**

---

**TESTAR towards 2025**

- Let the testing tool learn itself how to test
- Use different machine learning algorithms (action selection/oracles)
- Define more surrogate measures
- Learn from what the tool tests
- Show that surrogate measures work
- Extract models to aid exploratory testing
- Improve visualisation
- More formal testing theory
- Know better whether we have done well
- Reduce the human oracle cost
  - Automate as much of the oracle as possible
  - Make it as easy as possible for the tester

**Let us do some TESTAR!**
Virtual Machine

- Copy it onto your machine
- Install version 5.2.10 from virtualbox.org and the extensions
- Minimal requirements of your laptop:
  - 8GB of memory
  - A dual-core processor
  - 80GB of free disk space
  - Or connect to (5 people can connect)
  - testar.org:60001
  - ....
  - testar.org:60005

Hands-on guide


Getting started with TESTAR

Run testar.bat
The TESTAR modes

TESTAR modes

TESTAR SPY mode

TESTAR SPY mode

TESTAR SPY

TESTAR GENERATE mode

TESTAR SPY

TESTAR GENERATE mode

TESTAR SPY

TESTAR GENERATE mode
TESTAR REPLAY mode

Replay a previously tested sequence selected from the output directory.

TESTAR VIEW mode

View the steps of a previously tested sequence selected from the output directory.

The TESTAR protocol

- Allows you to adapt your test even more beyond the regular expression of the Dialogue
- Requires Java programming

How? Edit the protocol

The protocol editor
Follows the TESTAR flow

Connect to the SUT

3 different SUT connectors

- SUT connector is a setting that defines the way that TESTAR connects to the SUT.
- There are three ways:
  - COMMAND_LINE
  - SUT_PROCESS_NAME
  - SUT_WINDOW_TITLE
- The default is set to connect through COMMAND_LINE

COMMAND_LINE

- Start up the SUT by executing the given command line String
- In the text field you can type the String that represents the command that TESTAR will use to connect to the SUT

For example with the Calculator as a java jar

SUT_PROCESS_NAME

- Start up the SUT by connecting to a running process
- In the text field you can type the String that represents the name of the process
- Look up the name in for example the task manager

SUT_PROCESS_NAME

- Start up the SUT by connecting to a running process
- In the text field you can type the String that represents the name of the process
Connect to the SUT by connecting to a running process that has a specified windows title. In the text field you can type the String that represents the title of the window.

Adding Oracles

Oracles in Test

- ONLINE and OFFLINE oracles
- ONLINE: Give verdicts about current state of the GUI and the SUT
- We have automated:
  - Implicit oracles
    - Crashes
    - Hangs
    - Suspicious titles (regular expressions)
  - Other:
    - Domain-specific oracles

100% Automated online oracles

- Verdict oracle_crash (State state):
  ```java
  if (!state.get(IsRunning, false))
  return new Verdict("System crashed!");
  ```

- Verdict oracle_responsiveness (State state):
  ```java
  if (state.get(NotResponding, true))
  return new Verdict("System not responding!");
  ```

Oracle – Suspicious titles (under the hood)

```java
Verdict oracle_suspiciousTitles (State state) {
  // Get the regular expression from the Dialog
  String regEx = settings().get(SuspiciousTitles);
  // Search all widgets for suspicious titles
  for (Widget w : state) {
    String title = w.get(Title, "")
    if (title.matches(regEx))
      return new Verdict("Suspicious title!");
  }
}
```
In the protocol that can be edited through the TESTAR Dialog

Add any oracle by additional programming

3 ways to filter actions

1. **Regular expressions** action filter through test.settings or TESTAR Dialog
2. **Clickfilter** in SPY mode
3. **Programming** additional filters in deriveActions methods of the protocol.

Regular expression action filter

Clickfilter in SPY mode
Programming action filters

Visualizing test results

Offline oracles: Query the graph database

How to set it up for a specific SUT (at a client)

1) Planning Phase:
   a) Implementation of Test Environment: Recognize all the UI elements
   b) Anticipate and identify potential fault patterns: what errors do we want to find:
      - Detect crashes is for free and automatically
      - Suspicious output or messages is easy (regular expression)
      - Functionality as defined in the specifications (need oracle programming)

Integrating TESTAR into testing practice
How to set it up for a specific SUT (at a client)

2) Implementation Phase:
   a) Oracle Implementation
      - Implement the detection of the errors defined in the previous step.
   b) Action Definition Implementation
   c) Implementation of stopping criteria

3) Testing Phase: run the test to find errors in the SUT

TESTAR towards 2025

- Let the testing tool learn itself how to test
  - Use different machine learning algorithms (action selection/oracles)
  - Define more surrogate measures
- Learn from what the tool tests
  - Extract models to aid exploratory testing
  - Improve visualization
- More formal testing theory
  - Know better whether we have done well
- Reduce the human oracle cost
  - Automate as much as possible all other test tasks
  - Make it as easy as possible for the tester

- email: info@testar.org
- web: http://www.testar.org/
- LIKE IT on: http://facebook.com/tool.testar
- telephone/whatsapp: +34 690 917 971