6th UCAAT User Conference on Advanced Automated Testing

ETSI

Paris, 16-18 October 2018

Organizer: TESTING SOLUTIONS &SERVICES

Scriptless Test Automation through Graphical User Interface

Presented by Pekka Aho, Open Universiteit, NL

Open Universiteit
www.ou.nl

6th UCAAT User Conference on Advanced Automated Testing

ETSI

Paris, 16-18 October 2018

Organizer: TESTING SOLUTIONS & SERVICES

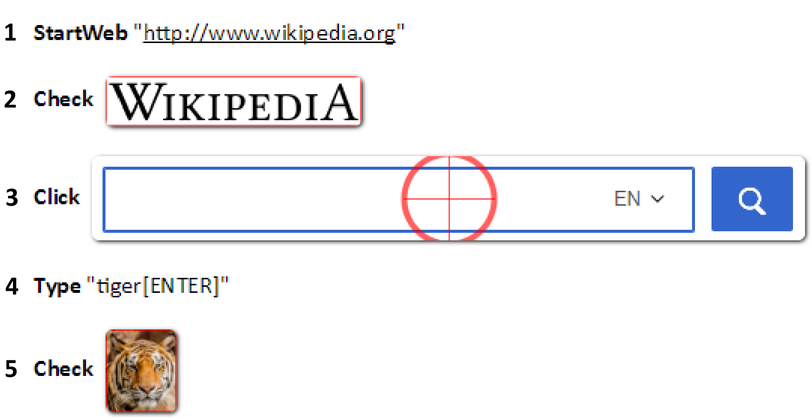
Test Automation through GUI

Scripted vs. Scriptless

Open Universiteit www.ou.nl

Scripted GUI testing – automated test execution

- Pre-defined sequence of test steps
  - Scripts usually manually created
  - Test oracles:
    - Assertions with expected values
    - Each check separately specified



1 StartWeb "http://www.wikipedia.org"

2 Check

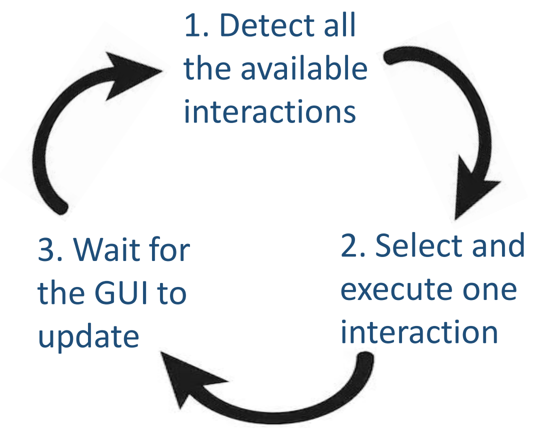
3 Click

4 Type "tiger[ENTER]"

5 Check

Scriptless GUI testing – automated GUI exploration

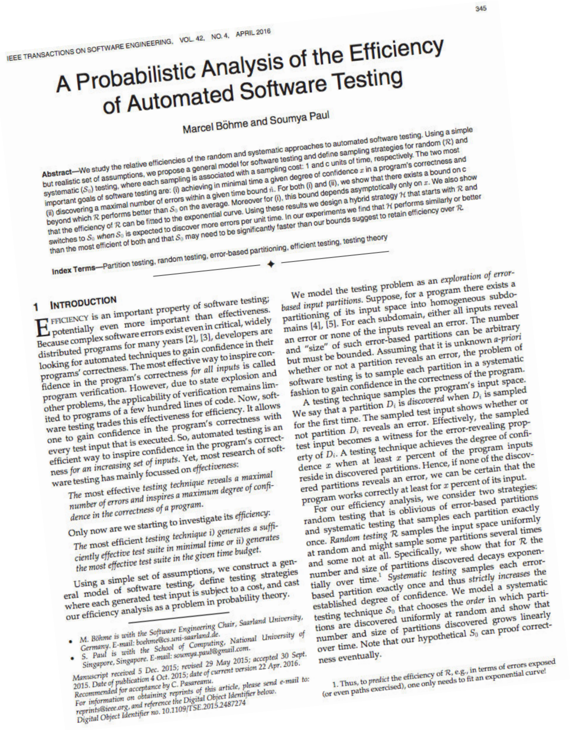
- Online / On-the-fly test generation
- Based on some level of randomness
  - Test oracles:
    - ”Free”: crashes, unresponsiveness, etc
    - Programmable: ”if text Error found...”



1. Detect all the available interactions

2. Select and execute one interaction

3. Wait for the GUI to update



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 42, NO. 4, APRIL 2016 345

A Probabilistic Analysis of the Efficiency of Automated Software Testing

Marcel Böhme and Soumya Paul

Abstract—We study the relative efficiencies of the random and systematic approaches to automated software testing. Using a simple but realistic set of assumptions, we propose a general model for software testing and define sampling strategies for random ($\mathcal{R}$) and systematic ($\mathcal{S}_0$) testing, where each sampling is associated with a sampling cost: 1 and c units of time, respectively. The two most important goals of software testing are: (i) achieving in minimal time a given degree of confidence x in a program's correctness and (ii) discovering a maximal number of errors within a given time bound $\hat{n}$. For both (i) and (ii), we show that there exists a bound on c beyond which $\mathcal{R}$ performs better than $\mathcal{S}_0$ on the average. Moreover for (i), this bound depends asymptotically only on x. We also show that the efficiency of $\mathcal{R}$ can be fitted to the exponential curve. Using these results we design a hybrid strategy $\mathcal{H}$ that starts with $\mathcal{R}$ and switches to $\mathcal{S}_0$ *when* $\mathcal{S}_0$ is expected to discover more errors per unit time. In our experiments we find that $\mathcal{H}$ performs similarly or better than the most efficient of both and that $\mathcal{S}_0$ may need to be significantly faster than our bounds suggest to retain efficiency over $\mathcal{R}$.

Index Terms—Partition testing, random testing, error-based partitioning, efficient testing, testing theory

1 INTRODUCTION

EFFICIENCY is an important property of software testing; potentially even more important than effectiveness. Because complex software errors exist even in critical, widely distributed programs for many years [2], [3], developers are looking for automated techniques to gain confidence in their programs' correctness. The most effective way to inspire confidence in the program's correctness *for all inputs* is called program verification. However, due to state explosion and other problems, the applicability of verification remains limited to programs of a few hundred lines of code. Now, software testing trades this effectiveness for efficiency. It allows one to gain confidence in the program's correctness with every test input that is executed. So, automated testing is an efficient way to inspire confidence in the program's correctness *for an increasing set of inputs*. Yet, most research of software testing has mainly focussed on *effectiveness*:

The most effective testing technique reveals a maximal number of errors and inspires a maximum degree of confidence in the correctness of a program.

Only now are we starting to investigate its *efficiency*:

The most efficient testing technique i) generates a sufficiently effective test suite in minimal time or ii) generates the most effective test suite in the given time budget.

Using a simple set of assumptions, we construct a general model of software testing, define testing strategies where each generated test input is subject to a cost, and cast our efficiency analysis as a problem in probability theory.

We model the testing problem as an *exploration of error-based input partitions*. Suppose, for a program there exists a partitioning of its input space into homogeneous subdomains [4], [5]. For each subdomain, either all inputs reveal an error or none of the inputs reveal an error. The number and "size" of such error-based partitions can be arbitrary but must be bounded. Assuming that it is unknown *a-priori* whether or not a partition reveals an error, the problem of software testing is to sample each partition in a systematic fashion to gain confidence in the correctness of the program.

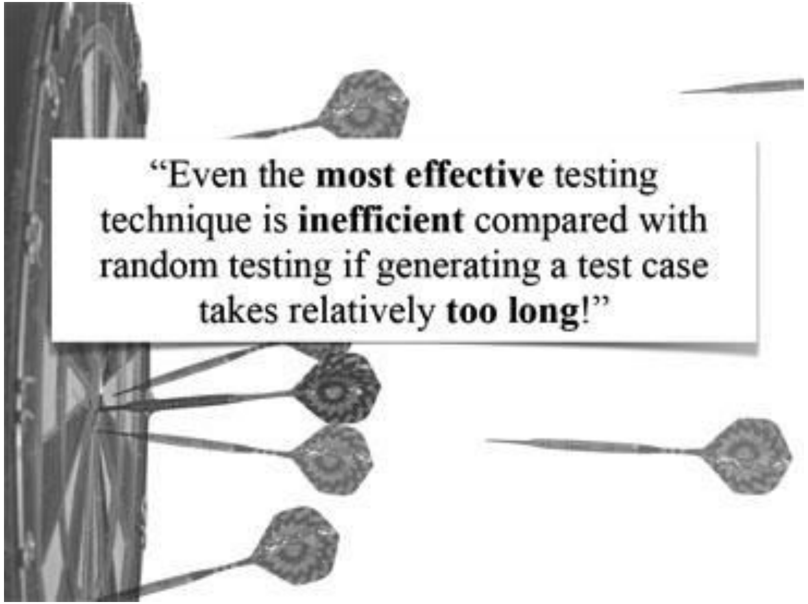
A testing technique samples the program's input space. We say that a partition D_i is *discovered* when D_i is sampled for the first time. The sampled test input shows whether or not partition D_i reveals an error. Effectively, the sampled test input becomes a witness for the error-revealing property of D_i. A testing technique achieves the degree of confidence x when at least x percent of the program inputs reside in discovered partitions. Hence, if none of the discovered partitions reveals an error, we can be certain that the program works correctly at least for x percent of its input.

For our efficiency analysis, we consider two strategies: random testing that is oblivious of error-based partitions and systematic testing that samples each partition exactly once. *Random testing* $\mathcal{R}$ samples the input space uniformly at random and might sample some partitions several times and some not at all. Specifically, we show that for $\mathcal{R}$ the number and size of partitions discovered decays exponentially over time.[1] *Systematic testing* samples each error-based partition exactly once and thus *strictly increases* the established degree of confidence. We model a systematic testing technique $\mathcal{S}_0$ that chooses the *order* in which partitions are discovered uniformly at random and show that number and size of partitions discovered grows linearly over time. Note that our hypothetical $\mathcal{S}_0$ can proof correctness eventually.

- *M. Böhme is with the Software Engineering Chair, Saarland University, Germany. E-mail: boehme@cs.uni-saarland.de.*
- *S. Paul is with the School of Computing, National University of Singapore, Singapore. E-mail: soumya.paul@gmail.com.*

Manuscript received 5 Dec. 2015; revised 29 May 2015; accepted 30 Sept. 2015. Date of publication 4 Oct. 2015; date of current version 22 Apr. 2016.
Recommended for acceptance by C. Pasareanu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2015.2487274

[1] Thus, to *predict* the efficiency of $\mathcal{R}$, e.g., in terms of errors exposed (or even paths exercised), one only needs to fit an exponential curve!



"Even the **most effective** testing technique is **inefficient** compared with random testing if generating a test case takes relatively **too long**!"

Scripted vs. Scriptless GUI testing

- Scripted
  - ❖Precise oracles
  - ❖Manual effort to create and maintain
- Scriptless
  - ❖Low maintenance
  - ❖General oracles, requires time to get coverage

- Scripted smoke tests and critical test scenarios
- Scriptless nightly testing for robustness and coverage

6th UCAAT User Conference on Advanced Automated Testing

ETSI

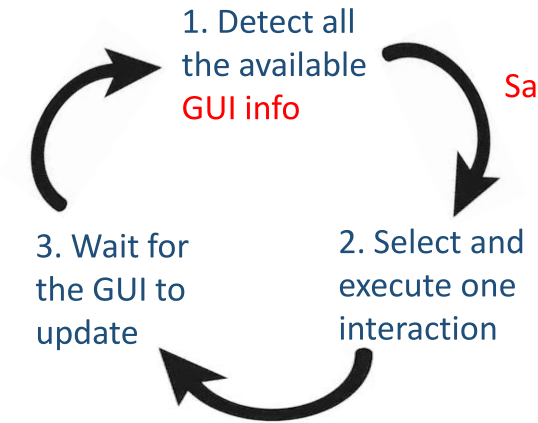
Paris, 16-18 October 2018

Organizer: TESTING SOLUTIONS &SERVICES

Scriptless Test Automation through GUI

Model Extraction

Scriptless GUI testing – GUI state model extraction



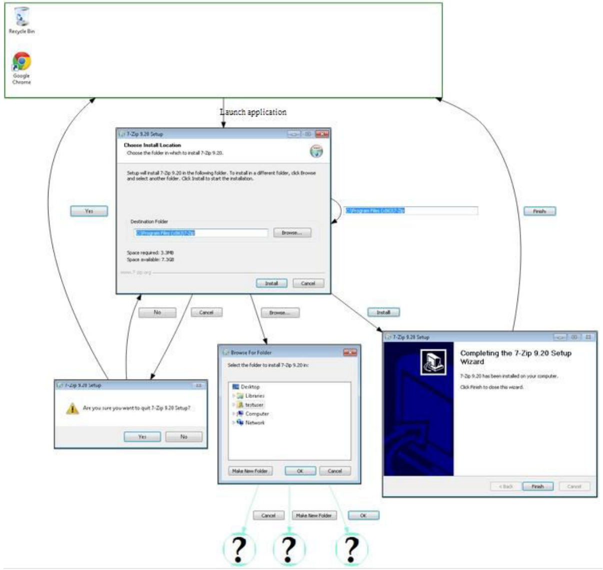
1. Detect all the available GUI info

Save as a GUI state

2. Select and execute one interaction

Save as a transition

3. Wait for the GUI to update



Choose Install Location

Choose the folder in which to install 7-Zip 9.20.

Setup will install 7-Zip 9.20 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\7-Zip

Space required: 3.3MB
Space available: 7.3GB

www.7-zip.org

Are you sure you want to quit 7-Zip 9.20 Setup?

Select the folder to install 7-Zip 9.20 in:

Completing the 7-Zip 9.20 Setup Wizard

7-Zip 9.20 has been installed on your computer.

Click Finish to close this wizard.

GUI state model extraction - challenges

- Abstraction
  - State-space explosion vs. ambiguous model
  - Could be application specific
- Manual elaboration of generated models
  - Preserving manual details when re-generating

Exploiting extracted GUI state models

- Automated documentation
- Analysis through visual inspection of models
  - Unspecified behavior
- Model-based testing
  - Manual elaboration (e.g. Test oracles)
- Automated change analysis by comparing GUI models of consequent versions
  - Report and visualize changes with screenshots

Model comparison for automated change analysis

- Reducing the need for scripted regression test cases
    - Reducing manually created scripts and maintenance effort
    - Increasing coverage
        - Covering also the improbable paths
        - Detecting all changes, not only assertions

6th UCAAT User Conference on Advanced Automated Testing

ETSI

TESTOMAT project

The Next Level of Test Automation

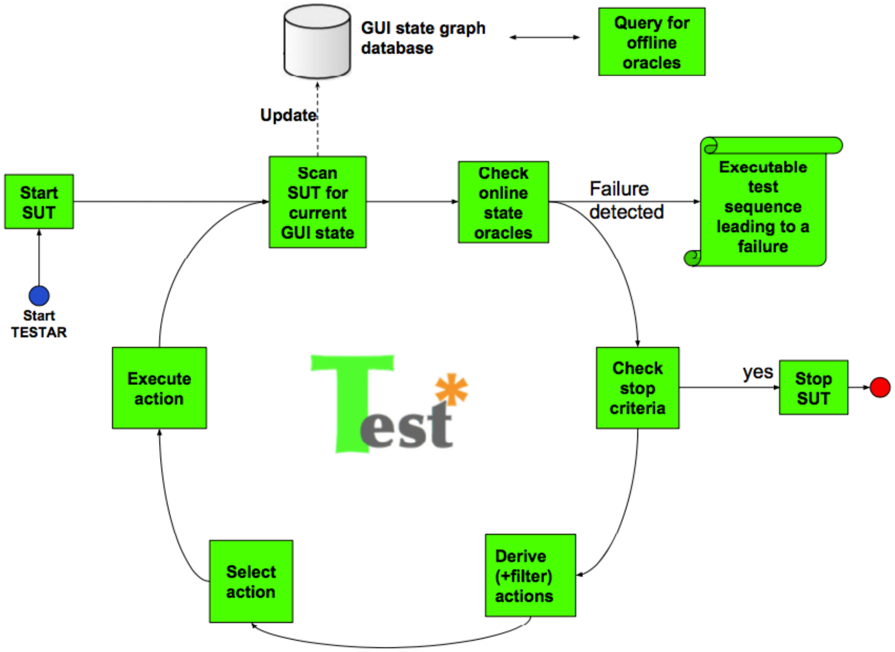
TESTOMAT PROJECT

The Next Level of Test Automation

- www.testomatproject.eu
- ITEA3 framework project, http://itea3.org/
- Industry-academia collaboration
- 34 partners from 6 countries

Test*

- testar.org
- Open source tool for scriptless GUI test automation
- Being extended / enhanced in TESTOMAT project



Scriptless GUI test automation in TESTOMAT

- Ongoing pilots with 3 partners
- Challenging industrial GUI apps
  - Tool has to be extended
- Pilots progressing slower than expected
  - Changes in CI pipeline
  - Other tool pilots at the same time
  - Some results still confidential