Automated smart test design
and its applications in
software transplantation, improvement and android testing

Mark Harman

Talk by Mark Harman
based on
PhD work by Ke Mao, Alex Marginean

Jointly supervised by
Yue Jia

University College London
COWs

CREST Open Workshop

Roughly one per month

Discussion based

Recorded and archived

http://crest.cs.ucl.ac.uk/cow/
COWs

CREST Open Workshop
Roughly one per month

Discussion based
Recorded and archived

http://crest.cs.ucl.ac.uk/cow/
COWs

CREST Open Workshop
Roughly one per month

Discussion based

Recorded and archived

http://crest.cs.ucl.ac.uk/cow/
COWs

http://crest.cs.ucl.ac.uk/cow/
COWs

#Total Registrations 1690
#Unique Attendees 721
#Unique Institutions 257
#Countries 44
#Talks 455

(Last updated on September 2016)

http://crest.cs.ucl.ac.uk/cow/
Search Based Optimization

Software Testing
In SBST we apply search techniques to search large search spaces, guided by a fitness function that captures natural counterparts as test objectives.

- Tabu Search
- Ant Colonies
- Hill Climbing
- Simulated Annealing
- Particle Swarm Optimization
- Genetic Algorithms
- Genetic Programming
- Greedy
- Estimation of Distribution Algorithms
- Random
- LP
Search Based Software Engineering

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures natural counterparts as test objectives.

Tabu Search  Ant Colonies  Particle Swarm Optimization
Hill Climbing  Genetic Algorithms
Simulated Annealing  Genetic Programming
Estimation of Distribution Algorithms  Greedy  LP  Random
SBSE Tutorial and Survey
SBSE Tutorial and Survey

SBSE Tutorial and Survey


SBSE Tutorial and Survey


google: SBSE tutorial

SBSE Tutorial and Survey


google: SBSE tutorial


google: SBSE survey
SBSE Tutorial and Survey


google: SBSE tutorial


google: SBSE survey
SBSE REPOSITORY

This page collects the work which addresses the software engineering problems using metaheuristic search optimisation techniques (i.e. Genetic Algorithms) into the Repository of Publications on Search Based Software Engineering.

- SBSE repository is maintained by Yuanzhu Zhang
- 1389 relevant publications are included
- Last updated on the 3 February 2005
- SBSE Authors on Google Scholar

Who's Who

The number of publications in the year from 1970 to 2001.

The ratio of all research fields that involved SBSE.

The ratio of publications number in the world countries.
Polynomial yearly rise in the number of papers Search Based Software Testing

\[ y = 0.0013x^4 - 0.061x^3 + 1.0008x^2 - 5.8636x + 10.443 \]
The changing ratio SBSE to SBST
Structural find tests to cover branches, statements & dataflow, etc.
Integration
Integration

find
best component
ordering
Temporal
Temporal

find worst case execution time
find 2-way, 3-way n-way interaction tests
Augment
find new tests from old tests
Regression
find good subsets and orders of tests
Functional
Mutation
State based
Model based
Black box
Failure Analysis
Security
Achievements, open problems and challenges for search based software testing

Mark Harman, Yue Jia and Yuanyuan Zhang
University College London, CREST Centre, London, UK

Abstract—Search Based Software Testing (SBST) formulates testing as an optimisation problem, which can be attacked using computational search techniques from the field of Search Based Software Engineering (SBSE). We present an analysis of the SBST research agenda, focusing on the open problems and challenges of testing non-functional properties, in particular a topic we call ‘Search Based Energy Testing’ (SBET). Multi-objective SBST and SBSE for Test Strategy Identification. We conclude with a vision of future tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FUTURE tools constitute an exciting challenge for the SBSE community that already could be within its reach.

I. INTRODUCTION

Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typically with large complex search spaces. Test objectives find natural counterparts as the fitness functions used by SBSE to guide automated search, thereby facilitating SBSE formulations of many (and diverse) testing problems. As a result, SBST has proved to be a widely applicable and effective way of generating test data, and optimising the testing process. However, there are many exciting challenges and opportunities that remain open for further research and development, as we will show in this paper.

It is widely believed that approximately half the budget spent on software projects is spent on software testing, and that no industry has a more complex, expensive, time-consuming, and error-prone process. In the last decade, as software testing has become less efficient, and as the need for increased quality assurance has increased, search-based software testing has emerged as a promising new approach to testing. A key component of search-based software testing is the use of optimisation techniques to automatically generate test cases. This approach has been shown to be effective in a number of domains, and has the potential to revolutionise software testing.

II. A BRIEF HISTORY OF SBST

Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 [51], and which only achieved more widespread acceptance and uptake many years later [38], [43], [100].

The first mention of software optimisation (of any kind) is almost certainly due to Ada Augusta Lovelace in 1842. Her English translation of the article (written in Italian by Menabrea), ‘Sketch of the Analytical Engine Invented by Charles Babbage’ includes seven entries, labelled ‘Note A’ to ‘Note G’ and initiated ‘A.A.L.’ Her notes constituted an article themselves (and occupied three quarters of the whole document). In these notes we can see perhaps the first recognition of the need for software optimisation and source code analysis and manipulation (a point argued in more detail elsewhere [44]).

“In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.” Excerpt from ‘Note D’.

The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually

Search Based Software Engineering (SBSE). We present an analysis of the SBST research agenda, focusing on the open problems and challenges of testing non-functional properties, in particular a topic we call ‘Search Based Energy Testing’ (SBET). Multi-objective SBST and SBSE for Test Strategy Identification. We conclude with a vision of future tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FUTURE tools constitute an exciting challenge for the SBSE community that already could be within its reach.

I. INTRODUCTION

Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces. Test objectives find natural counterparts as the fitness functions used by SBSE to guide automated search, thereby facilitating SBSE formulations of many (and diverse) testing problems. As a result, SBST has proved to be a widely applicable and effective way of generating test data, and optimising the testing process. However, there are many exciting challenges and opportunities that remain open for further research and development, as we will show in this paper.

It is widely believed that approximately half the budget spent on software projects is spent on software testing, and that no industry has a more complex, expensive, time-consuming, and error-prone process. In the last decade, as software testing has become less efficient, and as the need for increased quality assurance has increased, search-based software testing has emerged as a promising new approach to testing. A key component of search-based software testing is the use of optimisation techniques to automatically generate test cases. This approach has been shown to be effective in a number of domains, and has the potential to revolutionise software testing.

II. A BRIEF HISTORY OF SBST

Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 [51], and which only achieved more widespread acceptance and uptake many years later [38], [43], [100].

The first mention of software optimisation (of any kind) is almost certainly due to Ada Augusta Lovelace in 1842. Her English translation of the article (written in Italian by Menabrea), ‘Sketch of the Analytical Engine Invented by Charles Babbage’ includes seven entries, labelled ‘Note A’ to ‘Note G’ and initiated ‘A.A.L.’ Her notes constituted an article themselves (and occupied three quarters of the whole document). In these notes we can see perhaps the first recognition of the need for software optimisation and source code analysis and manipulation (a point argued in more detail elsewhere [44]):

“In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.” Excerpt from ‘Note D’.

The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually
Fig. 2: System Environment and Sub-Components of the Autonomous Parking System

Joachim Wegener and Oliver Bühler. GECCO 2004
### Table I

**Variables of Interest for the Prediction Models.**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Abbreviation</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fault in-flow</td>
<td>F. in-flow</td>
<td>Fault-inflow</td>
</tr>
<tr>
<td>2</td>
<td>No. of work packages planned for system integration</td>
<td>No. WP, PL, SI</td>
<td>Status rankings of WPs</td>
</tr>
<tr>
<td>3</td>
<td>No. of work packages delivered to system integration</td>
<td>No. WP, DEL, SI</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No. of work packages tested by system integration</td>
<td>No. WP Tested, SI</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>No. of faults slipping through to all of the testing phases</td>
<td>No. FST</td>
<td>FST</td>
</tr>
<tr>
<td>6</td>
<td>No. of faults slipping through to the unit testing</td>
<td>FST-Unit</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>No. of faults slipping through to the function testing</td>
<td>FST-Func</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>No. of faults slipping through to the integration testing</td>
<td>FST-Integ</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>No. of faults slipping through to the system testing</td>
<td>FST-Sys</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>No. of system test cases planned</td>
<td>No. System, TCs, PL</td>
<td>TC progress</td>
</tr>
<tr>
<td>11</td>
<td>No. of system test cases executed</td>
<td>No. System, TCs, Exec.</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>No. of interoperability test cases planned</td>
<td>No. IOT TCs, PL</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>No. of interoperability test cases executed</td>
<td>No. IOT TCs, Exec.</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>No. of network signaling test cases planned</td>
<td>No. NS TCs, PL</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>No. of network signaling test cases executed</td>
<td>No. NS TCs, Exec.</td>
<td></td>
</tr>
</tbody>
</table>

(c) Box plots of the residuals for each technique at the function testing phase.

Wasif Afzal, Richard Torkar, Robert Feldt and Greger Wikstrand. SSBSE 2010
Nikolai Tillmann, Jonathan de Halleux and Tao Xie. ASE 2014
Kiran Lakhotia, Mark Harman, and Hamilton Gross. I&ST 2013
EvoSuite automatically generates test cases for Java code. An excellent and high recommended tool.

Gordon Fraser and Andrea Arcuri. ESEC/FSE 2011
NEW KID: SAPIENZ
for fully-automated Android testing

Unfortunately, Facebook has stopped.

Report

OK
Does crash matter?

A crash is a fatal failure:
- Lost your data/progress
- Fatal for domains e.g., medical

In this work we report crashes, but

THE STATE OF THE ART

fully-automated Android testing

THE STATE OF THE ART

fully-automated Android testing

<table>
<thead>
<tr>
<th>Technique</th>
<th>Venue</th>
<th>Publicly Available</th>
<th>Box</th>
<th>Approach</th>
<th>Crash Report</th>
<th>Replay Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>AndroidRipper</td>
<td>ASE’12</td>
<td>Yes</td>
<td>Black</td>
<td>Model-based</td>
<td>Text</td>
<td>No</td>
</tr>
<tr>
<td>ACTEve</td>
<td>FSE’12</td>
<td>Yes</td>
<td>White</td>
<td>Program analysis</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>A’E</td>
<td>OOPSLA’13</td>
<td>Partially</td>
<td>Grey</td>
<td>Model-based</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>SwiftHand</td>
<td>OOPSLA’13</td>
<td>Yes</td>
<td>Black</td>
<td>Model-based</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>ORBIT</td>
<td>FASE’13</td>
<td>No</td>
<td>Grey</td>
<td>Model-based</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Dynodroid</td>
<td>FSE’13</td>
<td>Yes</td>
<td>Black</td>
<td>Random-based</td>
<td>Text, Image</td>
<td>Yes</td>
</tr>
<tr>
<td>PUMA</td>
<td>MobiSys’14</td>
<td>Yes</td>
<td>Black</td>
<td>Model-based</td>
<td>Text</td>
<td>Yes</td>
</tr>
<tr>
<td>EvoDroid</td>
<td>FSE’14</td>
<td>No</td>
<td>White</td>
<td>Search-based</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>SPAG-C</td>
<td>TSE’15</td>
<td>No</td>
<td>Black</td>
<td>Record-replay</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>MonkeyLab</td>
<td>MSR’15</td>
<td>No</td>
<td>Black</td>
<td>Trace mining</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>Thor</td>
<td>ISSTA’15</td>
<td>Yes</td>
<td>Black</td>
<td>Adverse conditions</td>
<td>Text</td>
<td>No</td>
</tr>
<tr>
<td>TrimDroid</td>
<td>ICSE’16</td>
<td>Yes</td>
<td>White</td>
<td>Program analysis</td>
<td>Text</td>
<td>Yes</td>
</tr>
<tr>
<td>CrashScope</td>
<td>ICST’16</td>
<td>No</td>
<td>Black</td>
<td>Combination</td>
<td>Text, Image</td>
<td>Yes</td>
</tr>
</tbody>
</table>

THE STATE OF THE ART

“Are we there yet? … Definitely NOT”

MOTIVATION EXAMPLE

monkey testing
SAPIENZ WORKFLOW

Ke Mao - Automated Mobile Testing: Dumb Monkeys, Smart Monkeys and Sapienz

APP SRC/APK

Instrumented APK

Android Device

Gene Interpreter

Test Generator

Test Replayer

States Logger

Multi-level Instrumenter

Decompiler

Static Strings

DB

Fitness Extractor

Report Generator

Crash Report

Coverage Report

Replay Video

Solutions (Test Suites)

Initialiser

GA

Select

Vary

Atomic Genes

Motif Genes

MotifCORE

SAPIENZ
SAPIENZ WORKFLOW

1. **SOURCE/APK** (SRC/APK)
2. **Instrumented APK**
3. **Multi-level Instrumenter**
4. **Decompiler**
5. **Static Strings**
6. **Android Device**
7. **States Logger**
8. **Database (DB)**
9. **Fitness Extractor**
10. **Gene Interpreter**
11. **Test Replayer**
12. **Test Generator**
13. **Initialiser**
14. **Gene Interpreter**
15. **Motif Genes**
16. **Test Replayer**
17. **Fitness Extractor**
18. **Evaluate**
19. **Select**
20. **Variant (Test Suites)**
21. **GA (Genetic Algorithm)**
22. **Report Generator**
23. **Crash Report**
24. **Coverage Report**
25. **Replay Video**
26. **Solutions (Test Suites)**

**Related Terms**:
- **Atomic Genes**
- **Motif Genes**
- **Multi-level Instrumenter**
- **Initialiser**
- **Fitness Extractor**
- **DB (Database)**
- **Gene Interpreter**
- **Test Generator**
- **Test Replayer**
- **Fitness Extractor**
- **Evaluate**
- **Select**
- **Variant (Test Suites)**
- **GA (Genetic Algorithm)**
- **Report Generator**
- **Crash Report**
- **Coverage Report**
- **Replay Video**
- **Solutions (Test Suites)**
EMULATOR MODE
REAL DEVICE MODE

System-level Testing
Mobile App Testing
Event-driven App Testing
Automated Exploratory Testing
THREE EVALUATIONS

### COVERAGE

- **Sapienz**: 53%
- **Dynodroid**: 44%
- **Monkey**: 48%

### FAULTS

- **Sapienz**: 558 bugs
- **Dynodroid**: 7 bugs
- **Monkey**: 13 bugs

### LENGTH

- **Sapienz**: 15,305
- **Dynodroid**: 161
- **Monkey**: 149

### Statistical Significance

- 7

### Top 1000 Google Play Apps

- 14/27 confirmed bugs
Automated Android Testing

- custom fit tailored into your development process
give us the app; get detailed test report

- service fit repeatable fault revelation

- non flakey minimised debugging effort

- minimised length test thought leadership; open source systems

- commitment to our community
How much do you really trust testing?

Refuse: I didn’t write it!

Wait: Until I prove it correct

Experiment: Generate tests
How much do you *really trust* testing?

Experiment: Generate tests
How much do you really trust testing?

Experiment: Generate tests
Genetic Improvement of Programs

Bowtie2 → Sensitivity Analysis → GP → Improved Bowtie2

Non-functional property Test harness

Test data → Fitness

70 times faster
30+ interventions
HC clean up: 7
slight semantic improvement

W. B. Langdon and M. Harman
Optimising Existing Software with Genetic Programming. TEC 2015
Genetic Improvement of Programs

- Sensitivity Analysis
- GP
  - Test data
  - Fitness
- Non-functional property Test harness
Genetic Improvement of Programs

Cuda → Sensitivity Analysis → GP → Cuda Improved

Test data
Fitness
Non-functional property Test harness

7 times faster
updated for new hardware
automated updating

W. B. Langdon and M. Harman
Genetically Improved CUDA C++ Software, EuroGP 2014
Inter version transplantation

Sensitivity Analysis → GP

Test data → Fitness

Non-functional property Test harness
Inter version transplantation

Justyna Petke, Mark Harman, William B. Langdon and Westley Weimer
Using Genetic Improvement & Code Transplants to Specialise a C++ program to a Problem Class (EuroGP’14)

GECCO Humie silver medal
Real world cross system transplantation

- Sensitivity Analysis
- GP
- Test data
- Fitness
- Non-functional property Test harness
Real world cross system transplantation

Automated Software Transplantation
Earl Barr, Mark Harman, Yue Jia, Alexandru Marginean and Justyna Petke
ISSTA 2015. Distinguished paper award.

3x5 = 15 experiments
12 were successful
Automated Software Transplantation
Earl Barr, Mark Harman, Yue Jia, Alexandru Marginean and Justyna Petke
ISSTA 2015.
Memory speed trade offs

Sensitivity Analysis → GP

Test data
Fitness

Non-functional property test harness

Search Based Software Engineering - SBSE
Mark Harman, UCL CREST
Memory speed trade offs

Improve execution time by 12% or achieve a 21% memory consumption reduction

Fan Wu, Westley Weimer, Mark Harman, Yue Jia and Jens Krinke
Deep Parameter Optimisation
Conference on Genetic and Evolutionary Computation (GECCO’15).
Reducing energy consumption

- Sensitivity Analysis
- Non-functional property Test harness
- Fitness
- Test data
- GP

Mark Harman, UCL CREST
Reducing energy consumption

Energy consumption can be reduced by as much as 25%

Bobby R. Bruce Justyna Petke Mark Harman
Reducing Energy Consumption Using Genetic Improvement
Conference on Genetic and Evolutionary Computation (GECCO’15).
Grow and graft new functionality

Non-functional property Test harness

Sensitivity Analysis

GP

Test data

Fitness

?
Grow and graft new functionality

Mark Harman, Yue Jia and Bill Langdon,
Babel Pidgin: SBSE can grow and graft entirely new functionality into a real world system
Symposium on Search-Based Software Engineering SSBSE 2014. (Challenge track)
AUTOMATED TESTING IS MATURING

- **Coverage**
  - Sapienz: 53%
  - Dynodroid: 44%
  - Monkey: 48%

- **Faults**
  - Sapienz: 104
  - Dynodroid: 7
  - Monkey: 41

- **Length**
  - Sapienz: 15,305
  - Dynodroid: 7
  - Monkey: 10

558 bugs

14/27 confirmed bugs

Ke Mao – Automated Mobile Testing: Dumb Monkeys, Smart Monkeys and Sapienz