Achievements, Open Problems and Challenges for Search based Software Testing

Mark Harman
Joint work with Yue Jia and Yuanyuan Zhang

University College London
Achievements, open problems and challenges for search based software testing

Mark Harman, Yue Jia and Yanyuan 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 refer to as ‘Search Based Energy Testing’ (SBET). Multi-objective SBST and SBET for Test Strategy Identification. We conclude with a vision of future tasks, which would automatically find tests, fix them and verify the fix. We explain why we think such future SBST tasks 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 [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typically by large complex search spaces [36]. 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 existing 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 therefore, it is not surprising that perhaps a similar proportion of papers in the software engineering literature are concerned with software testing. We report an updated literature analysis from which we observe that approximately half of all SBSE papers are SBST papers, a figure little changed since the last thorough publication audit (for papers up to 2009), which found 54% of SBSE papers concerned SBST [36]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [15], [85], [120]. Therefore, rather than attempting another survey, we provide an analysis of SBST research trends, focusing on open challenges and areas for future work and development.

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 1970's but not formally established as a field of study in its own right until 2001 [21], and which only achieved more widespread acceptance and uptake many years later [18], [43], [100].

The first mention of software optimisation of any kind is almost certainly due to Ada Augusta Lovelace in 1842. Her English language 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 (to point argued in more detail elsewhere [44]).

"To 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." Extract from 'Note E'.

The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both software testing and software verification. The first use of optimisation techniques in software testing and verification probably dates back to the seminal PhD thesis by James King [67], who used automatic symbolic execution to capture path conditions, solved using linear programming. The first formulation of the test input space as a search space probably dates back seven years earlier to 1962, when a Cobol test data generation tool was introduced by Sauder [103]. Sauder formulates the test generation problem as one of finding test inputs from a search space, though the search algorithm is random search, making this likely to be the first paper on Random Test Data Generation. Sauder’s work is also significant because it introduces the idea of constraints to capture path conditions, although these constraints are manually defined and not automatically constructed.

There is a paper accompany this keynote
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 SBST for Test Strategy Identification. We conclude with a vision of FIP/VERIFY tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FIP/VERIFY 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 [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces [58]. Test objectives find natural counterparts in the SBST framework, as do the many other techniques that are familiar from search in artificial intelligence.

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 language translation of the article (written in Italian by Menahrae), ‘Sketch of the Analytical Engine Invented by Charles Babbage’ includes seven entries, labelled ‘Note A’ to ‘Note G’ and initialed ‘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]):

“By almost every computation a great variety of
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 SBST for Test Strategy Identification. We conclude with a vision of FEPIVERIFY tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such 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 [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces [58]. Test objectives find natural counterparts therein as fitness functions to guide search.

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 language 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
Yue Jia
Technical work
and considerable help with slides

... and he’s here in Graz too
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 1347
#Unique Attendees 623
#Unique Institutions 232
#Countries 42
#Talks 372

(Last updated on January 31, 2015)

http://crest.cs.ucl.ac.uk/cow/
What is SBST

- Search Based Optimization
- Software Testing
What is SBST

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, Random, Estimation of Distribution Algorithms,
What is SBSE
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
Search-based software engineering

Mark Harman a,⁎, Bryan F. Jones b,1

aDepartment of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
bSchool of Computing, University of Glamorgan, Pontypridd, CF37 1DL, UK

Abstract

This paper claims that a new field of software engineering research and practice is emerging: search-based software engineering. The paper argues that software engineering is ideal for the application of metaheuristic search techniques, such as genetic algorithms, simulated annealing and tabu search. Such search-based techniques could provide solutions to the difficult problems of balancing competing (and sometimes inconsistent) constraints and may suggest ways of finding acceptable solutions in situations where perfect solutions are either theoretically impossible or practically infeasible.

In order to develop the field of search-based software engineering, a reformulation of classic software engineering problems as search problems is required. The paper briefly sets out key ingredients for successful reformulation and evaluation criteria for search-based software engineering. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Software engineering; Metaheuristic; Genetic algorithm
“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.”

Extract from ‘Note D’.
In this shot paper, Turing suggested the use of manually constructed assertions and we can find the origins of both software testing and software verification.
Sauder formulates the test generation problem as one of finding test inputs from a search space, though the search algorithm is random search, making this likely to be the first paper on **Random Test Data Generation**.
The seminal PhD thesis by James King

James King used automated symbolic execution to capture path conditions, solved using linear programming.
“We therefore considered various alternatives that would not be subject to this limitation. The most promising of these alternatives appears to be a conjugate gradient algorithm (‘hill climbing’ program) that seeks to minimise a potential function constructed from the inequalities.”
At about the same time, Miller and Spooner were also experimenting with optimisation-based approaches for generating test data (which they refer to as ‘test selection’ in the sense that they ‘select’ from the input space, which, in the more recent literature we would refer to as ‘test data generation’).
It appears that SBST research lay dormant for approximately a decade until the work of Korel, which introduced a practical test data generation approach, the Alternating Variable Method (AVM), based on hill climbing.
The first use of genetic algorithms for software engineering problems is usually attributed also to the field of SBST, with the work of Xanthakis et al., who introduced a genetic algorithm to develop whole test suites.
The first suggestion of search as a universal approach to Software Engineering

The first use of genetic algorithm to develop whole test suites
1992

Application of genetic algorithms to software testing

Authors: S Xanthakis, C Ellis, C Skourias, A Le Gali, S Katsikas, K Karapoulis
Publication date: 1992/12/7
Journal: Proceedings of the 5th International Conference on Software Engineering and its Applications
Pages: 625-636
Total citations: Cited by 156

The first suggestion of search as a universal approach to Software Engineering
The first use of genetic algorithm to develop whole test suites
1992

Application of genetic algorithms to software testing

Authors: S. Xanthakis, C. Ellis, C. Stourlas, A. La Gall, S. Kalivas, K. Karapavlis
Publication date: 1992/12/27
Journal: Proceedings of the 6th International Conference on Software Engineering and Its Applications
Pages: 625-634
Total citations: Cited by 156

Formally established as a filed of study with SBSE
2001

Search-based software engineering
Analysis of Trends in SBST

The data is taken from the SBSE

Careful human-based update

100% precision and recall
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
SBST papers at ICST

Reformulating Branch Coverage as a Many-Objective Optimization Problem
Annibale Panichella, Fitsum Meshesha Kifetew and Paolo Tonella

Behind an Application Firewall, Are We Safe from SQL Injection Attacks?
Dennis Appelt, Cu D. Nguyen, Lionel Briand

Exploring Test Suite Diversification and Code Coverage in Multi-Objective Test Case Selection
Debajyoti Mondal, Hadi Hemmati, and Stephane Durocher

Guided Test Generation for Finding Worst-Case Stack Usage in Embedded Systems
Tingting Yu and Myra B. Cohen

Re-using Generators of Complex Test Data
Simon Poulding and Robert Feldt

U-Test: Evolving, Modelling and Testing Realistic Uncertain Behaviours of Cyber-Physical Systems
Shaukat Ali and Tao Yue

History-Based Test Case Prioritization for Black Box Testing using Ant Colony Optimization
Tadahiro Noguchi, Hironori Washizaki, Yoshiaki Fukazawa, Atsutoshi Sato and Kenichiro Ota

Combining Minimization and Generation for Combinatorial Testing
Itai Segall, Rachel Tzoref-Brill and Aviad Zlotnick.
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 interaction tests
Augment
find new tests from old tests
Regression
Regression

find good subsets and orders of tests
State based
Black box
Achievements, Open Problems and Challenges for SBST

Security
Achievements, Open Problems and Challenges for SBST

Mark Harman
SBST’s Industrial Applications and Tools

Fig. 2: System Environment and Sub-Components of the Autonomous Parking System

Joachim Wegener and Oliver Bühler. GECCO 2004
SBST’s Industrial Applications and Tools

Wasif Afzal, Richard Torkar, Robert Feldt and Greger Wikstrand. SSBSE 2010
SBST’s Industrial Applications and Tools

Nikolai Tillmann, Jonathan de Halleux and Tao Xie. ASE 2014
SBST Public Tools

AUSTIN applied to real-world embedded automotive industry: Daimler, B&M Systemtechnik. Recommended for testing C.

Kiran Lakhotia, Mark Harman, and Hamilton Gross. I&ST 2013
SBST Public Tools

EvoSuite automatically generates test cases for Java code. An excellent and highly recommended tool.

Gordon Fraser and Andrea Arcuri. ESEC/FSE 2011
ICST’15 Achievements, Open Problems and Challenges for SBST

Mark Harman, Yue He and Yanqun Zhang

University College London, CRICK Centre, London, UK

1. Introduction

Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing [2, 10]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), treated by large complex search spaces [9]. Test objectives that material components in the design process are used by SBSE in a guided automation of the search process. As a result, SBST has proved to be a widely applicable and effective way of generating test data and operating the testing process. However, there are many existing 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 project is spent on software testing, and therefore, it is not surprising that perhaps a similar proportion of papers in the software engineering literature are concerned with software testing. We report an updated literature analysis from which we observe that approximately half of all SBSE papers have been published in search spaces since 2000. Other papers covering SBSE (2000-2010) include [2, 10]. Although excellent and detailed surveys of the SBSE literature can be found elsewhere [9], [10], [9], [10], the present work, rather than attempting another survey, provide a new analysis of SBST research trends, focusing on open challenges and areas for future work and development.

This keynote paper was written by Mark Harman in the 11th RM International Conference on Software Testing, Verification and Validation (ICST 2015) for the 2015 edition of the keynote paper series. It is the only keynote paper in the ICST proceedings that includes an analysis section.

Analysis

More details in the keynote paper in your ICST proceedings
SBST’s Challenges

We need to extend SBST to test non-functional properties. In particular, we need more work on Search Based Energy Testing (SBET).

We need Search Based Test Strategy Identification (SBTSI).

We need more work on multi-objective test data generation techniques (MoSBaT).
SBST’s Challenges

We need to extend SBST to test non-functional properties. In particular, we need more work on Search Based Energy Testing (SBET).

We need Search Based Test Strategy Identification (SBTSI).

We need more work on multi-objective test data generation techniques (MoSBaT).
SBST’s Challenges

We need to extend SBST to test non-functional properties. In particular, we need more work on Search Based Energy Testing (SBET).

We need Search Based Test Strategy Identification (SBTSI).

We need more work on multi-objective test data generation techniques (MoSBaT).
SBST’s Challenges

We need to extend SBST to test non-functional properties. In particular, we need more work on Search Based Energy Testing (SBET).

We need Search Based Test Strategy Identification (SBTSI).

We need more work on multi-objective test data generation techniques (MoSBaT).

Annibale Panichella, Fitsum Meshesha Kifetew and Paolo Tonella

Reformulating Branch Coverage as a Many-Objective Optimization Problem
ICST’15
Achievements, Open Problems and Challenges for SBST

Mark Harman, Yoo Jin and Yongyan Zhang
University College London, CREST Center, London, UK

SBST for Non-Functional Properties

II. A BRIEF HISTORY OF SBST

Since the first paper on SBST is also likely to be the first paper on SBST, 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 1950s but not formally established as a field of study in its own right until 2001 [5], and which only achieved more widespread acceptance and uptake many years later [14], [15], [18].

The first mention of software optimization (of any kind) is almost certainly due to Ada Augusta Lovelace in 1842. The English-language translation of the article written in Italian by Menabrea, “Sketch of the Analytical Engine Invented by Charles Babbage” includes seven anxious, labelled Note A to “Note C” and included A.N.I.D. Barrows constituted an article themselves (and occupied three quarters of the whole document). In those notes we can see perhaps the first recognition of the need for software optimisation and some early attempts to optimise a piece of code in more detail early on [40].

“...almost every computation a great variety of arrangements for the execution of the process 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.” Extract from “Note D”.

The introduction of the idea of software testing is probably due to Turing [13], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both software testing and software verification. The first use of verification techniques in software testing and verification probably dates back to the seminal PhD thesis by James King [87] who used automated symbolic execution to capture path conditions, solving using linear programming. The first description of the test input space as a search space probably dates back seven years earlier to 1962, when a Nobel test data generation tool was introduced by Sundar [103]. Sundar formulates the test generation problem as one of finding test inputs from a search space. Though the search algorithm in random search, making the likely to be the first paper on Random Test Data Generation. Sundar’s work is also significant because it introduces the idea of constraints to capture path conditions, although these constraints are manually defined and not automatically generated.
SBST for Non-Functional Properties

The Categories of Non-Functional Properties from 1996 to 2007

- Execution time: 15 (43%)
- Safety: 4 (11%)
- Usability: 7 (20%)
- QoS: 2 (6%)

The Categories of Non-Functional Properties from 1996 to 2014

- Execution time: 21 (25%)
- Robustness: 10 (12%)
- Scalability: 5 (6%)
- Efficiency: 8 (10%)
- Usability: 10 (12%)
- Security: 13 (15%)
- Safety: 8 (9%)
- Flexibility: 3 (4%)
- Energy consumption: 1 (1%)
- Availability: 1 (1%)

Tiny distribution of work

Tiny distribution of work
Search based Energy Test (SBET)
Uh oh.
Search based Energy Test (SBET)

A smartphone could consume more energy per year than a medium-sized refrigerator.
Search based Energy Test (SBET)

A smartphone could consume more energy per year than a medium-sized refrigerator

IT energy consumption rose 3% in 3 years
Search based Energy Test (SBET)

Measure energy consumption as a fitness function

**Efficiency:** will need to consider many different test cases
Search based Energy Test (SBET)

**Efficiency**: will need to consider many different test cases

**Coarse Granularity**: Energy consumed per run overall
Search based Energy Test (SBET)

**Efficiency**: will need to consider many different test cases

**Coarse Granularity**: Energy consumed per run overall

**Hawthorne Effect**: Instrumentation may affect energy consumed
Search based Energy Test (SBET)

**Efficiency**: will need to consider many different test cases

**Coarse Granularity**: Energy consumed per run overall

**Hawthorne Effect**: Instrumentation may affect energy consumed
39th COW - Measuring, Testing and Optimising Computational Energy Consumption
39th COW - Measuring, Testing and Optimising Computational Energy Consumption
Search Based Test Strategy Identification (SBTSI)

Move from finding **specific inputs** to finding **strategies for finding inputs**

- General CIT Solution
- Unknown CIT Problems

**A hyperheuristic SBTSI for CIT**

- A co-evolutionary SBTSI for Mutation testing
A hyperheuristic SBSTI for CIT

CIT Solutions

- AETG
- IPOG
- GA
- Simulated Annealing
- Tabu
- Hill Climbing
A hyperheuristic SBSTI for CIT

CIT Solutions
- AETG
- IPOG
- Simulated Annealing
- Tabu
- Hill Climbing

CIT Problem Characteristics
- Specific structures
- Unconstrained problems
- Constrained problems
- Weighted problems
A hyperheuristic SBSTI for CIT

CIT Solutions

- AETG
- IPOG
- GA
- Simulated Annealing
- Tabu
- Hill Climbing

CIT Problem Characteristics

- Specific structures
- unconstrained problems
- constrained problems
- Weighted problems

SBSTI
A hyperheuristic SBSTI for CIT

CIT Solutions

- AETG
- IPOG
- GA
- Simulated Annealing
- Tabu
- Hill Climbing

CIT Problem Characteristics

- Specific structures
- unconstrained problems
- constrained problems
- Weighted problems

SBSTI
A hyperheuristic SBSTI for CIT

CIT Solutions
- AETG
- IPOG
- Simulated Annealing
- Tabu
- Hill Climbing

CIT Problem Characteristics
- Specific structures
- Unknown CIT Problems
- Unconstrained problems
- Constrainted problems
- Weighted problems

SBSTI
A hyperheuristic SBSTI for CIT

CIT Solutions

- AETG
- IPOG
- GA
- Simulated Annealing
- Tabu
- Hill Climbing

CIT Problem Characteristics

- Specific structures
- Unknown CIT Problems
- Unconstrained problems
- Weighted problems
- Constrained problems
A hyperheuristic SBSTI for CIT

CIT Solutions

- General CIT Solution
- AETG
- IPOG
- Simulated Annealing
- Tabu
- Hill Climbing

CIT Problem Characteristics

- Unknown CIT Problems
- Specific structures
- Unconstrained problems
- Constrainted problems
- Weighted problems

ICST’15
Achievements, Open Problems and Challenges for SBST
## A co-evolutionary approach for SBSTI

<table>
<thead>
<tr>
<th>Predator</th>
<th>Prey</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>Bugs</td>
</tr>
</tbody>
</table>
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI
A co-evolutionary approach for SBSTI

Test data

Higher order mutants
Multi-Objective Search Based Testing (MOSBAT)

increasingly prevalent regression testing was early adopter

e.g. Yoo and Harman: ISSTA 2007
Multi-Objective Search Based Testing (MOSBAT)
Multi-Objective Search Based Testing (MOSBAT)

test case generation is still mostly single objective
Multi-Objective Search Based Testing (MOSBAT)
Multi-Objective Search Based Testing (MOSBAT)

Multi-objective Understanding:
Debug Security policies
SBST’s Challenges

We need to extend SBST to test non-functional properties. In particular, we need more work on **Search Based Energy Testing (SBET)**.

We need **Search Based Test Strategy Identification (SBTSI)**.

We need more work on **multi-objective test data generation techniques (MoSBaT)**.
Let me ask you something ...
Genetic Improvement: searching for improving modifications guided by testing
Genetic Improvement of Programs

1. Bowtie2 → Sensitivity Analysis
2. Sensitivity Analysis → GP
3. GP → Test data
4. Test data → Fitness
5. Fitness → 70 times faster
6. 70 times faster → 30+ interventions
7. 30+ interventions → HC clean up: 7
8. HC clean up: 7 → slight semantic improvement

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

ICST’15 Achievements, Open Problems and Challenges for SBST
**Genetic Improvement of Programs**

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

GI Programs

GP

Sensitivity Analysis
Genetic Improvement of Programs

- Sensitivity Analysis
  - 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)

Multi-donor transplant
Specialized for CIT
17% faster

GECCO Humie silver medal
Real world cross system transplantation

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

Doner feature

Sensitivity Analysis

GP

Test data

Fitness

Host feature

Non-functional property Test harness

Successfully autotransplanted new functionality and passed all regression tests for 12 out of 15 real world systems

Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke
Memory speed trade offs

- Sensitivity Analysis
- Non-functional property Test harness
- Test data
- Fitness
- GP
Fan Wu, Westley Weimer, Mark Harman, Yue Jia and Jens Krinke
Deep Parameter Optimisation
Conference on Genetic and Evolutionary Computation (GECCO’15), To appear
Memory speed trade offs

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

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

ICST'15 Achievements, Open Problems and Challenges for SBST
Mark Harman
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), To appear
Grow and graft new functionality

- Sensitivity Analysis
- Test data
- Fitness
- Non-functional property Test harness
- GP
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)
The GISMSE challenge:
Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs.

Mark Harman*, William B. Langdon†, Yue Jia‡, David R. White*, Andrea Arcuri*, John A. Clark
†CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
‡School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
*Simula Research Laboratory, P. O. Box 134, 3928 Lysestad, Norway.
Department of Computer Science, University of York, Daresmore Lane, York, YO10 5HD, UK.

ABSTRACT
Optimising programs for non-functional properties such as speed, size, throughput, power consumption and bandwidth can be demanding: pity the poor programmer who is asked to cater for them all at once! We set out an alternate vision for a new kind of software development environment inspired by recent results from Search Based Software Engineering (SBSE). Given an input program that satisfies the functional requirements, the proposed programming environment will automatically generate a set of candidate program implementations, all of which share functionality, but each of which offers an alternative non-functional trade-off. The software designer navigates this diverse Pareto surface of candidate implementations, gaining insight into the trade-offs and selecting solutions for different platforms and environments, thereby stretching beyond the reach of current compiler technologies. Rather than having to focus on the details required to manage complex, inter-related and conflicting non-functional trade-offs, the designer is thus freed to explore, to understand, to control and to decide rather than to construct.

Categories and Subject Descriptors
D.2 [Software Engineering]

General Terms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.

Keywords
SBSE, Search Based Optimisation, Compilation, Non-functional Properties, Genetic Programming, Pareto Surface.

1. INTRODUCTION
Humans find it hard to develop systems that balance many competing and conflicting non-functional objectives. Even meeting a single objective, such as execution time, requires automated support in the form of compiler optimisation. However, though most compilers can optimise compiled code for both speed and size, the programmer may find them selves making arbitrary choices when such objectives are in conflict with one another.

Furthermore, speed and size are but two of many objectives that the next generation of software systems will have to consider. There are many others such as bandwidth, throughput, response time, memory consumption and resource access. It is unrealistic to expect an engineer to decide, up-front, on the precise weighting that they attribute to each such non-functional property, nor for the engineer even to know what might be achievable in some unfamiliar environment to which the system may be deployed.

Emergent computing application paradigms require systems that are not only reliable, compact and fast, but which also optimise many different competing and conflicting objectives such as response time, throughput and consumption of resources (such as power, bandwidth and memory). As a result, operational objectives (the so-called non-functional properties of the system) are becoming increasingly important and uppermost in the minds of software engineers.

Software developers cannot be expected to optimally balance these multiple competing constraints and may even potentially valuable solutions should they attempt to do so. Why should they have to? How can a programmer assess (at code writing time) the behaviour of their code with regard to non-functional properties on a platform that may not yet have been built?

To address this concern we propose a development environment that distinguishes between functional and non-functional properties. In this environment, the functional properties remain the preserve of the human designer, while the optimisation of non-functional properties is left to the machine. That is, the choice of the non-functional properties to be considered will remain a decision for the human software designer.
Pareto Front
each circle is a program found by a machine
different non functional properties have different pareto program fronts
Failed Test Cases
Why can’t functional properties be optimisation objectives?
Optimisation
Optimisation

2.5 times faster but failed 1 test case?
Optimisation

double the battery life but failed 2 test cases?
Isn’t testing all about searching?

Searching for test cases
Searching for test application orders
Searching for patches
Searching for better programs guided by tests
Genetic Improvement
Isn’t testing all about searching?

Search for test cases
Searching for test application orders
Searching for patches

Search for better programs guided by tests

Genetic Improvement
Picture Copyrights

http://beatles.wikia.com/wiki/The_Beatles_Wiki