ICST 2015

8th IEEE International Conference on Software Testing, Verification and Validation

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 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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

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 automated 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[1], 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 FIFIVERIFY tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FIFIVERIFY tools constitute an exciting challenge for the SBSE community that already could be within its reach.**

I. Introduction

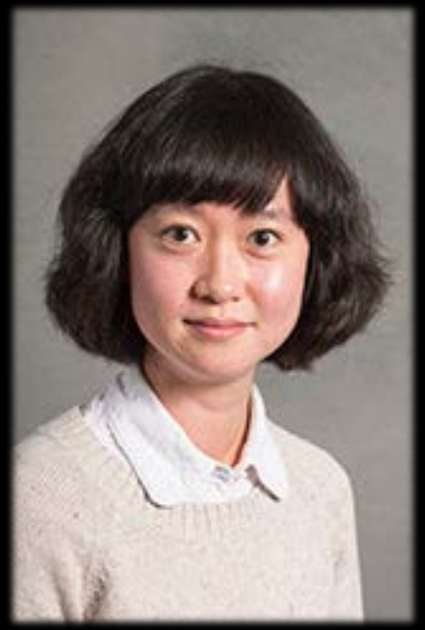
Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces [58]. Test objectives find natural counterparts as the [illegible] fun[illegible] [illegible] by [illegible]E to gui[illegible] automa[illegible]

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 Menabrae), '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]):

"*In almost every computation a great variety of*



Yuanyuan Zhang

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[1], 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 FIFIVERIFY tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FIFIVERIFY tools constitute an exciting challenge for the SBSE community that already could be within its reach.**

I. Introduction

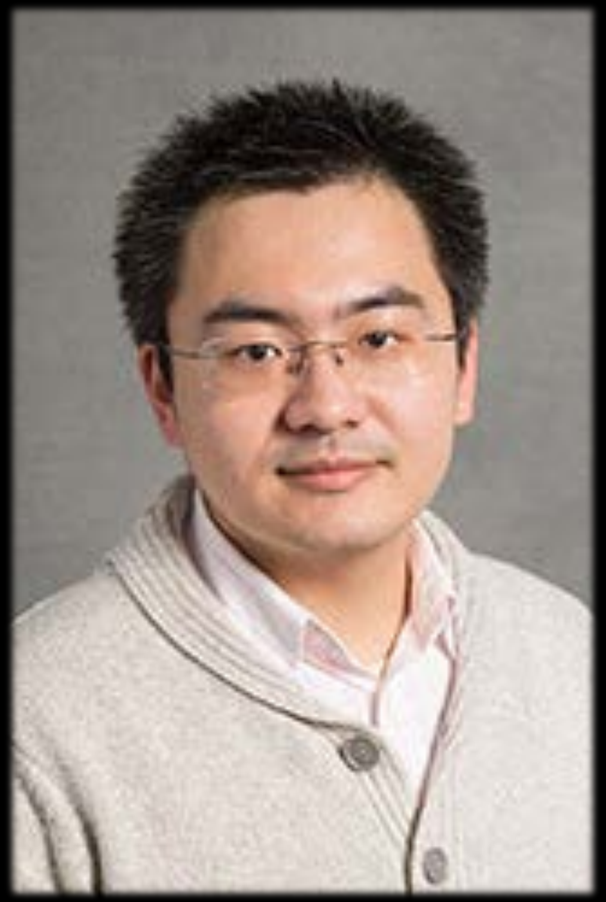
Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces [58]. Test objectives find natural counterparts as the [illegible] fun[illegible] by [illegible] to gui[illegible] automa[illegible]

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 Menabrae), '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]):

"*In almost every computation a great variety of*



Yue Jia
Technical work
and considerable help with slides

… and he’s here in Graz too


Madame Tussaud's
Sherlock Holmes Museum
20 mins walk
British Museum
UCL
Marble Arch
Eros
Nelson's Column
Covent Garden Market
RoyalCourts of Justice
St. Paul's
National Gallery
Tate Modern
Globe Theatre
London Eye
Westminster Abbey
National History Museum
House of Parliament

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/

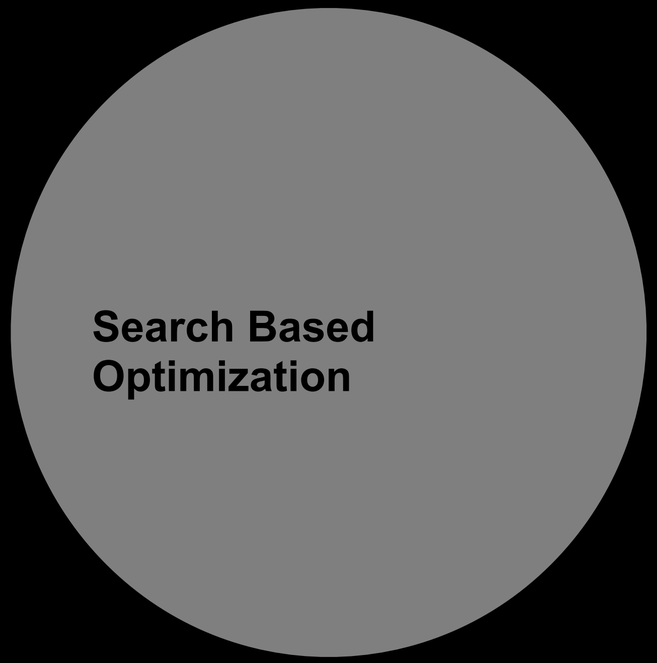
COWs

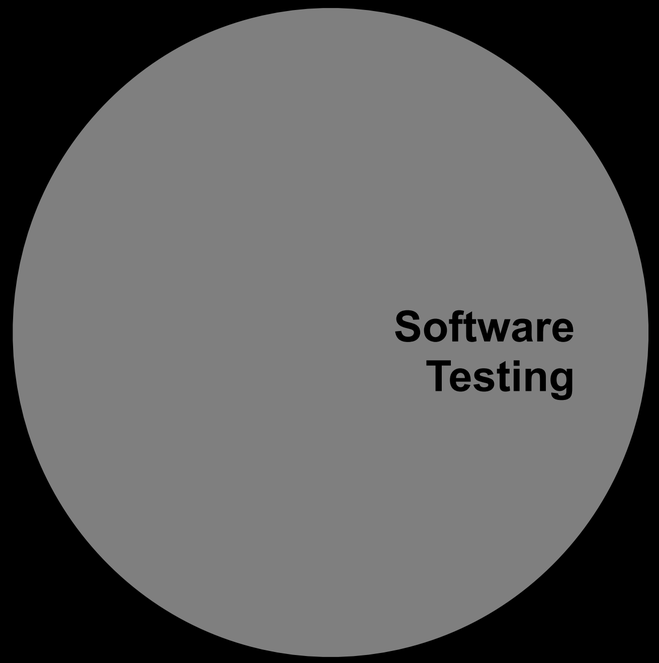
#Total Registrations 1347
#Unique Attendees 623
#Unique Institutions 232
#Countries 42
#Talks 372

(Last updated on January 31, 2015)

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

What is SBST





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
Particle Swarm Optimization
Genetic Algorithms
Hill Climbing
Genetic Programming
Simulated Annealing
Random
Greedy
LP
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
Genetic Algorithms
Hill Climbing
Genetic Programming
Simulated Annealing
Random
Greedy
LP
Estimation of Distribution Algorithms

History of

ELSEVIER

Information and Software Technology 43 (2001) 833–839

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.com/locate/infsof

Search-based software engineering

Mark Harman[a,*], Bryan F. Jones[b,1]

[a]Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex, UB8 3PH, UK
[b]School of Computing, University of Glamorgan, Pontypridd, CF37 1DL, UK

Abstract

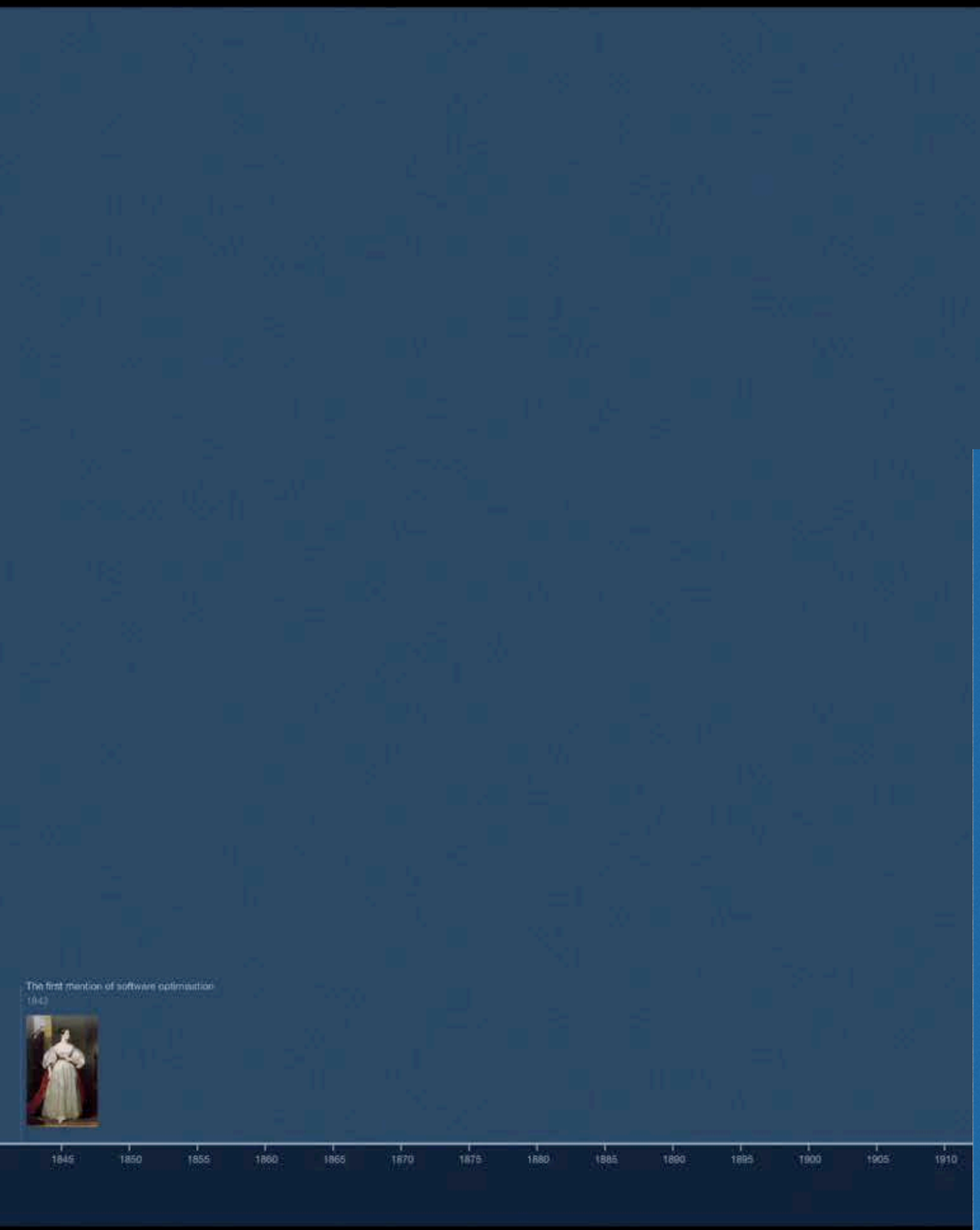
This paper claims that a new field of software engineering research and practice is emerging: search-based software engineering. The paper argues that software engineering is ideal for the application of metaheuristic search techniques, such as genetic algorithms, simulated annealing and tabu search. Such search-based techniques could provide solutions to the difficult problems of balancing competing (and some times 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.

Keywords: Software engineering; Metaheuristic; Genetic algorithm

1842

2001



"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**."

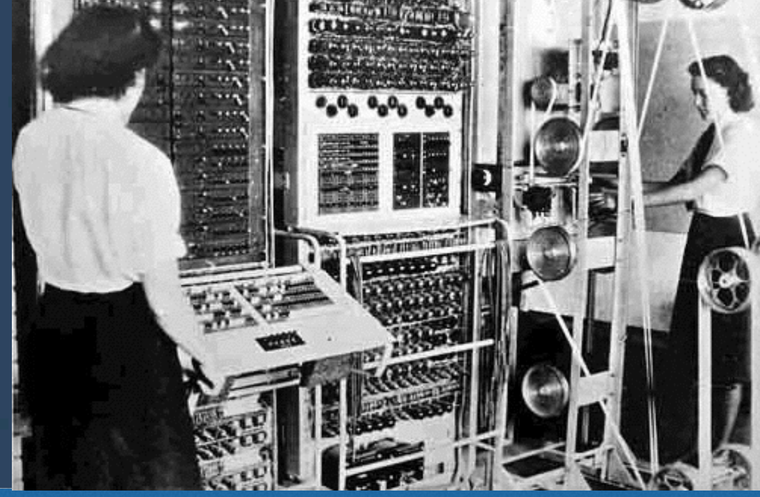
Extract from 'Note D'.

The first mention of software optimisation

1842

1845

1850

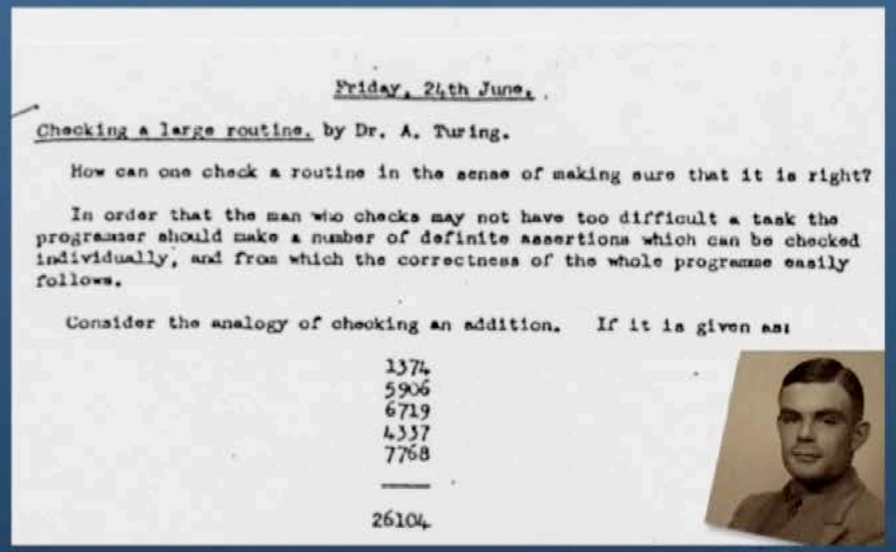


Checking a large routine
by Dr. A. Turing

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**.

The introduction of the idea of software testing and verification

1949



Friday, 24th June.

Checking a large routine, by Dr. A. Turing.

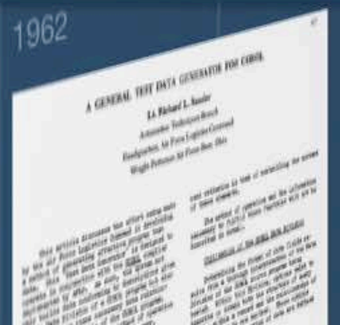
How can one check a routine in the sense of making sure that it is right?

In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole programme easily follows.

Consider the analogy of checking an addition. If it is given as:

1374
5906
6719
4337
7768
———
26104

1962



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**.

1990

Formulation of the test input space as a search space

1962

317

A GENERAL TEST DATA GENERATOR FOR COBOL

Lt. Richard L. Sauder
Automation Techniques Branch
Headquarters, Air Force Logistics Command
Wright-Patterson Air Force Base, Ohio

This article discusses the effort being made by the Air Force Logistics Command in developing a method of generating effective program test data. This "Test Data Generator" is designed to operate in conjunction with the COBOL compiler implemented by AFLC. As such, the system not only builds data conforming to descriptions given in the Data Division of a COBOL program but also places in these items necessary data relationships to test the logic of the COBOL program. Both the utilization and the method of operation of the system are discussed in this paper.

Introduction

One of the major underdeveloped areas that still exists in the development of programming techniques is that of insuring adequate checkout of programs before release for use. Often the logical paths within a program are highly complex. The effort to insure that each of these is functionally correct can be prohibitive - so much so that only the most obvious and most frequently employed segments of the program are tested thoroughly.

nent criterion is that of controlling the content of these elements.

The method of operation and the information necessary to fulfill these functions will now be described in detail.

Utilization of the COBOL Data Division

Determining the format of data fields results from a thorough interpretation of the Data Division of the COBOL source program being tested. Within this Division, options exist to describe in detail both the structure of every element within a record and the relationships of these elements to one another. Those options which affect the generation of data are defined as follows:

1. Name
2. Level number to establish the relationship of one unit of data to others.
3. Size in number of characters.
4. Class indicating the type of data, i.e., alphabetic, numeric, alphanumeric.
5. Usage to establish the number system in

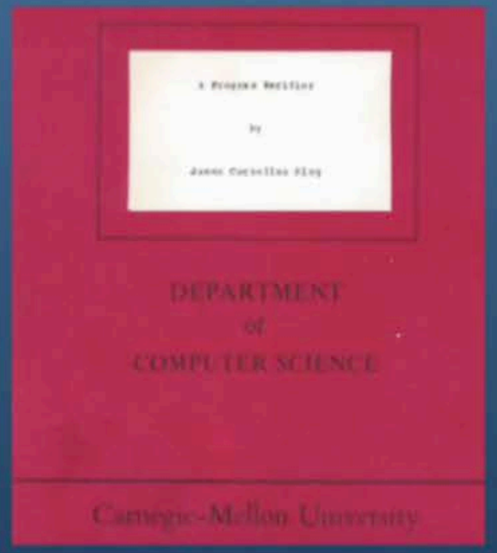
1976

The seminal PhD thesis
by James King

James King used automated symbolic execution to capture path conditions, solved using **linear programming**

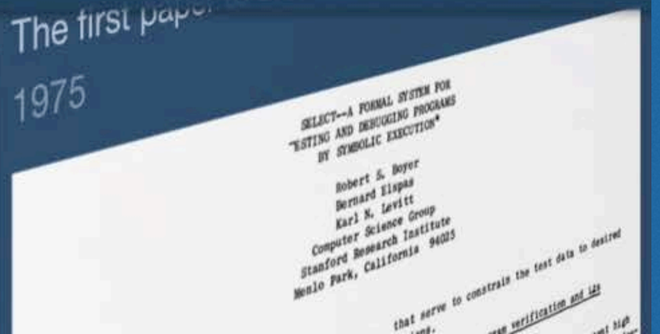
The first use of optimisation techniques in software testing and verification

1969



The first paper to use a meta-heuristic search technique

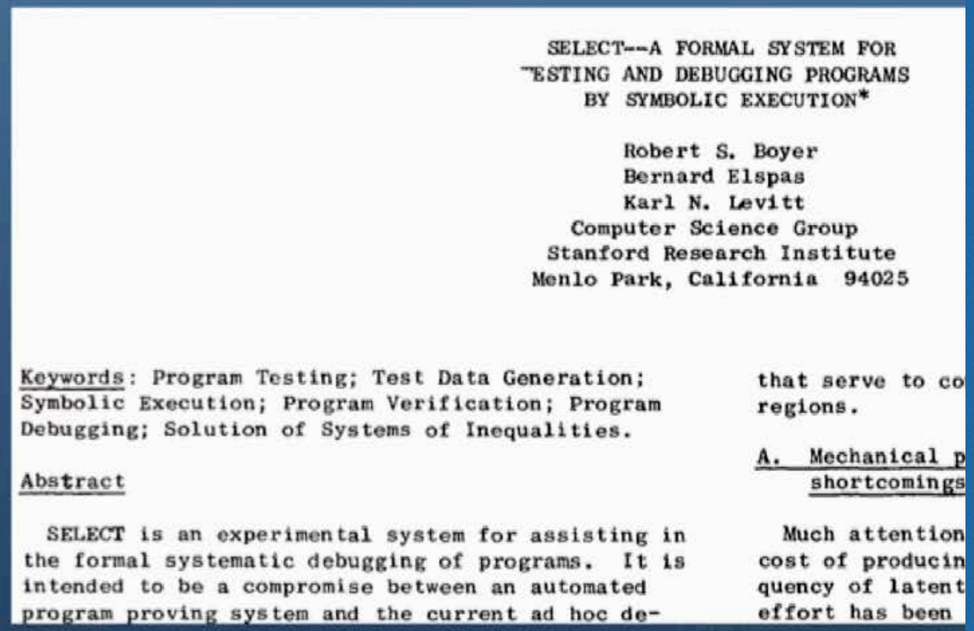
1975



"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."

The first paper to use a meta-heuristic search technique

1975



SELECT--A FORMAL SYSTEM FOR
TESTING AND DEBUGGING PROGRAMS
BY SYMBOLIC EXECUTION*

Robert S. Boyer
Bernard Elspas
Karl N. Levitt
Computer Science Group
Stanford Research Institute
Menlo Park, California 94025

Keywords: Program Testing; Test Data Generation; Symbolic Execution; Program Verification; Program Debugging; Solution of Systems of Inequalities.

Abstract

SELECT is an experimental system for assisting in the formal systematic debugging of programs. It is intended to be a compromise between an automated program proving system and the current ad hoc de-

that serve to co
regions.

A. Mechanical p
shortcomings

Much attention
cost of producin
quency of latent
effort has been

1990

Automated Software Test Data Gen

BOGDAN KOREL

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’).

Application of genetic algorithms to software testing

Authors: S Xanthakis, C Ellis, C Skourlas, A Le Gall, S Katsikas, K Karagiannis

Publication date: 1992/12/7

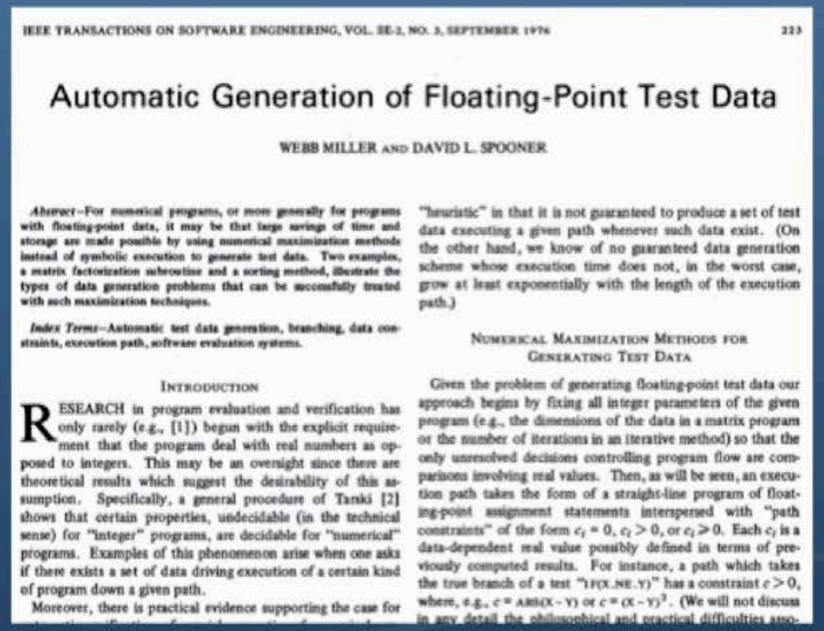
Journal: Proceedings of the 5th International Conference on Sof...

Pages: 625-636

Total citations: Cited...

The use of optimisation-based approach for 'test selection'

1976



IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-2, NO. 3, SEPTEMBER 1976 223

Automatic Generation of Floating-Point Test Data

WEBB MILLER AND DAVID L. SPOONER

Abstract—For numerical programs, or more generally for programs with floating-point data, it may be that large savings of time and storage are made possible by using numerical maximization methods instead of symbolic execution to generate test data. Two examples, a matrix factorization subroutine and a sorting method, illustrate the types of data generation problems that can be successfully treated with such maximization techniques.

Index Terms—Automatic test data generation, branching, data constraints, execution path, software evaluation systems.

Introduction

RESEARCH in program evaluation and verification has only rarely (e.g., [1]) begun with the explicit requirement that the program deal with real numbers as opposed to integers. This may be an oversight since there are theoretical results which suggest the desirability of this assumption. Specifically, a general procedure of Tarski [2] shows that certain properties, undecidable (in the technical sense) for "integer" programs, are decidable for "numerical" programs. Examples of this phenomenon arise when one asks if there exists a set of data driving execution of a certain kind of program down a given path.

Moreover, there is practical evidence supporting the case for

"heuristic" in that it is not guaranteed to produce a set of test data executing a given path whenever such data exist. (On the other hand, we know of no guaranteed data generation scheme whose execution time does not, in the worst case, grow at least exponentially with the length of the execution path.)

Numerical Maximization Methods for Generating Test Data

Given the problem of generating floating-point test data our approach begins by fixing all integer parameters of the given program (e.g., the dimensions of the data in a matrix program or the number of iterations in an iterative method) so that the only unresolved decisions controlling program flow are comparisons involving real values. Then, as will be seen, an execution path takes the form of a straight-line program of floating-point assignment statements interspersed with "path constraints" of the form $c_i = 0$, $c_i > 0$, or $c_i \geq 0$. Each c_i is a data-dependent real value possibly defined in terms of previously computed results. For instance, a path which takes the true branch of a test "IF(X.NE.Y)" has a constraint $c > 0$, where, e.g., $c = \text{ABS}(X - Y)$ or $c = (X - Y)^2$. (We will not discuss in any detail the philosophical and practical difficulties asso-

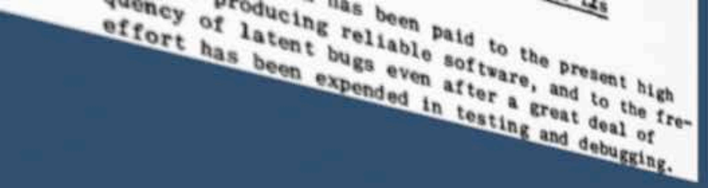
1985

1980

1975

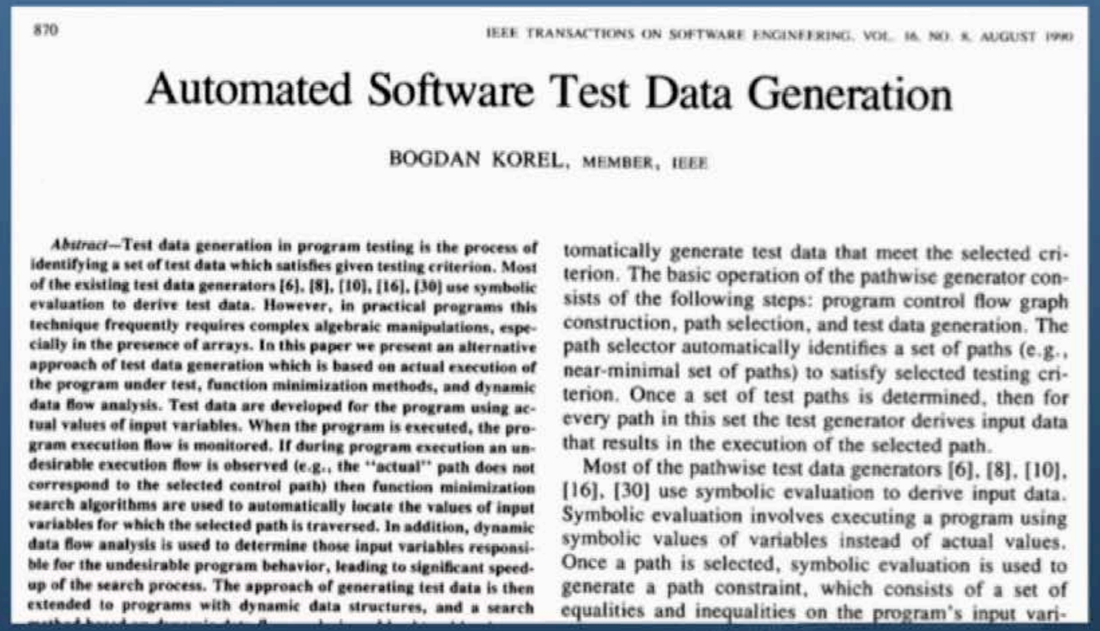
70

It appears that SBST research lay dormant for at 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.



A practical test data generation approach based on A

1990


870 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 16, NO. 8, AUGUST 1990

Automated Software Test Data Generation

BOGDAN KOREL, MEMBER, IEEE

Abstract—Test data generation in program testing is the process of identifying a set of test data which satisfies given testing criterion. Most of the existing test data generators [6], [8], [10], [16], [30] use symbolic evaluation to derive test data. However, in practical programs this technique frequently requires complex algebraic manipulations, especially in the presence of arrays. In this paper we present an alternative approach of test data generation which is based on actual execution of the program under test, function minimization methods, and dynamic data flow analysis. Test data are developed for the program using actual values of input variables. When the program is executed, the program execution flow is monitored. If during program execution an undesirable execution flow is observed (e.g., the "actual" path does not correspond to the selected control path) then function minimization search algorithms are used to automatically locate the values of input variables for which the selected path is traversed. In addition, dynamic data flow analysis is used to determine those input variables responsible for the undesirable program behavior, leading to significant speed-up of the search process. The approach of generating test data is then extended to programs with dynamic data structures, and a search

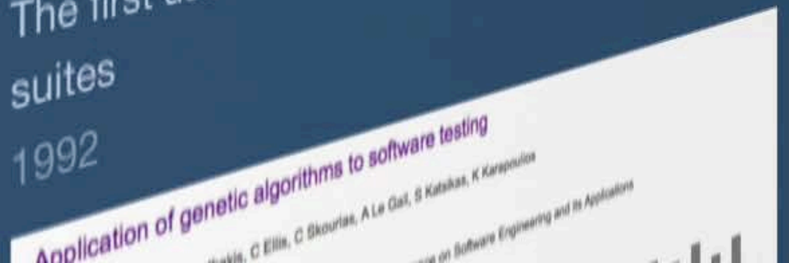
tomatically generate test data that meet the selected criterion. The basic operation of the pathwise generator consists of the following steps: program control flow graph construction, path selection, and test data generation. The path selector automatically identifies a set of paths (e.g., near-minimal set of paths) to satisfy selected testing criterion. Once a set of test paths is determined, then for every path in this set the test generator derives input data that results in the execution of the selected path.

Most of the pathwise test data generators [6], [8], [10], [16], [30] use symbolic evaluation to derive input data. Symbolic evaluation involves executing a program using symbolic values of variables instead of actual values. Once a path is selected, symbolic evaluation is used to generate a path constraint, which consists of a set of equalities and inequalities on the program's input vari-

The first use of genetic algorithm to develop whole

suites

1992



Windows 3.1

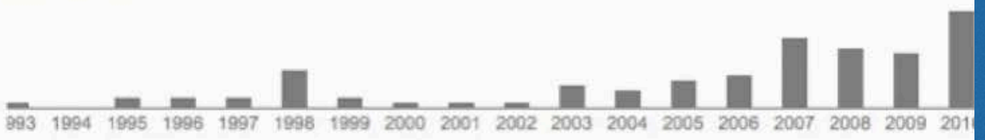
The **first use of genetic algorithms** for software engineering problems is usually attributed also to the field of SBST, with the work of Xanthakis et al., who introduced a genetic algorithm to develop whole test suites.

The first use of genetic algorithm to develop whole test suites

1992

Application of genetic algorithms to software testing

Authors	S Xanthakis, C Ellis, C Skourlas, A Le Gall, S Katsikas, K Karapoulios
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



d approach for 'test selection'

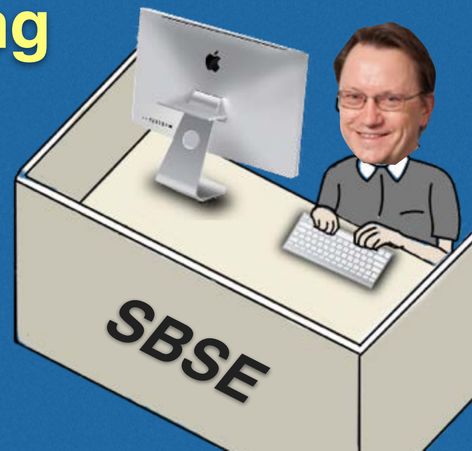
Formally

2001

Search-based software engineering

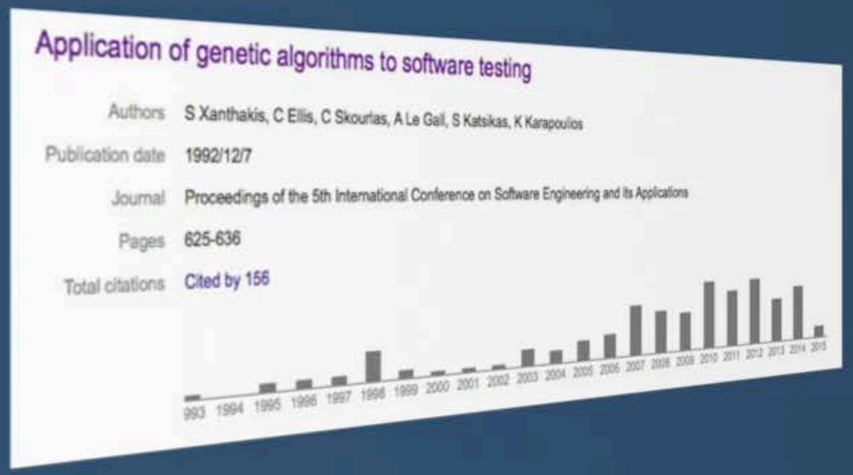
Mark Harman, Bryan F. Jones

The first suggestion of **search as a universal approach to Software Engineering**



The first use of genetic algorithm to develop whole test suites

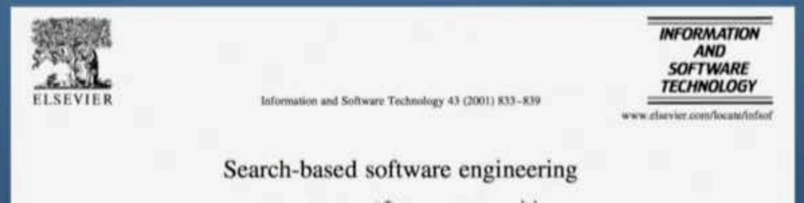
1992



d approach for 'test selection'

Formally established as a filed of study with SBSE

2001

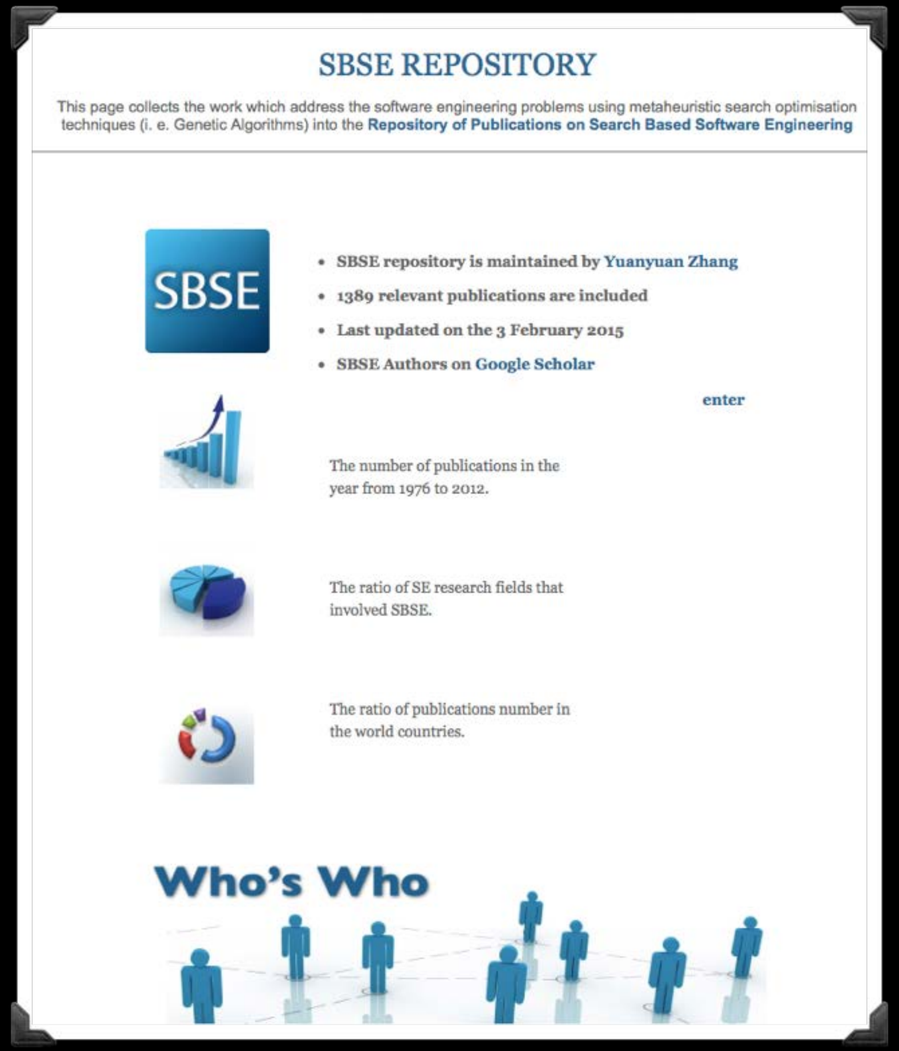


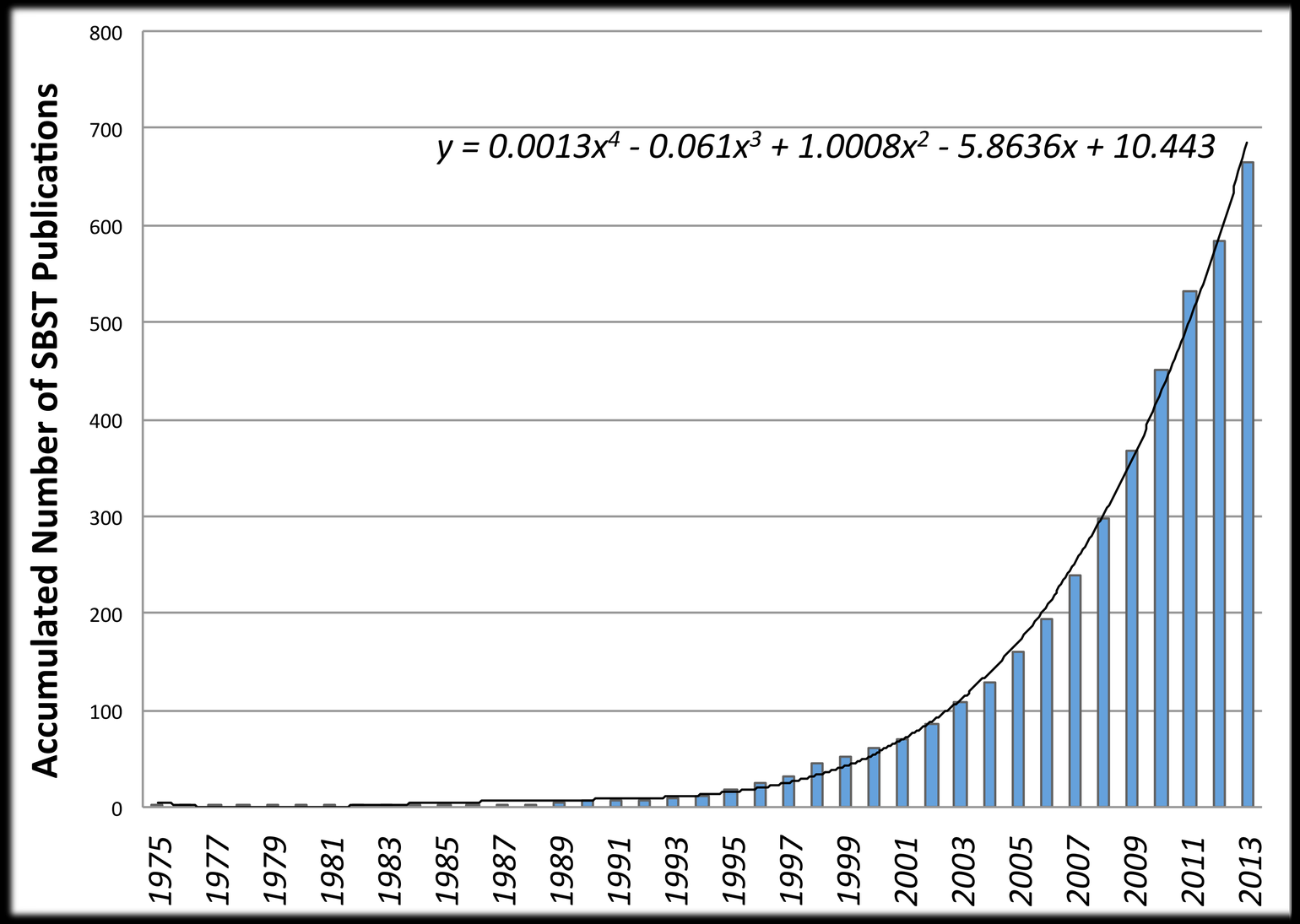
Analysis of Trends in SBST

The data is taken from the SBSE

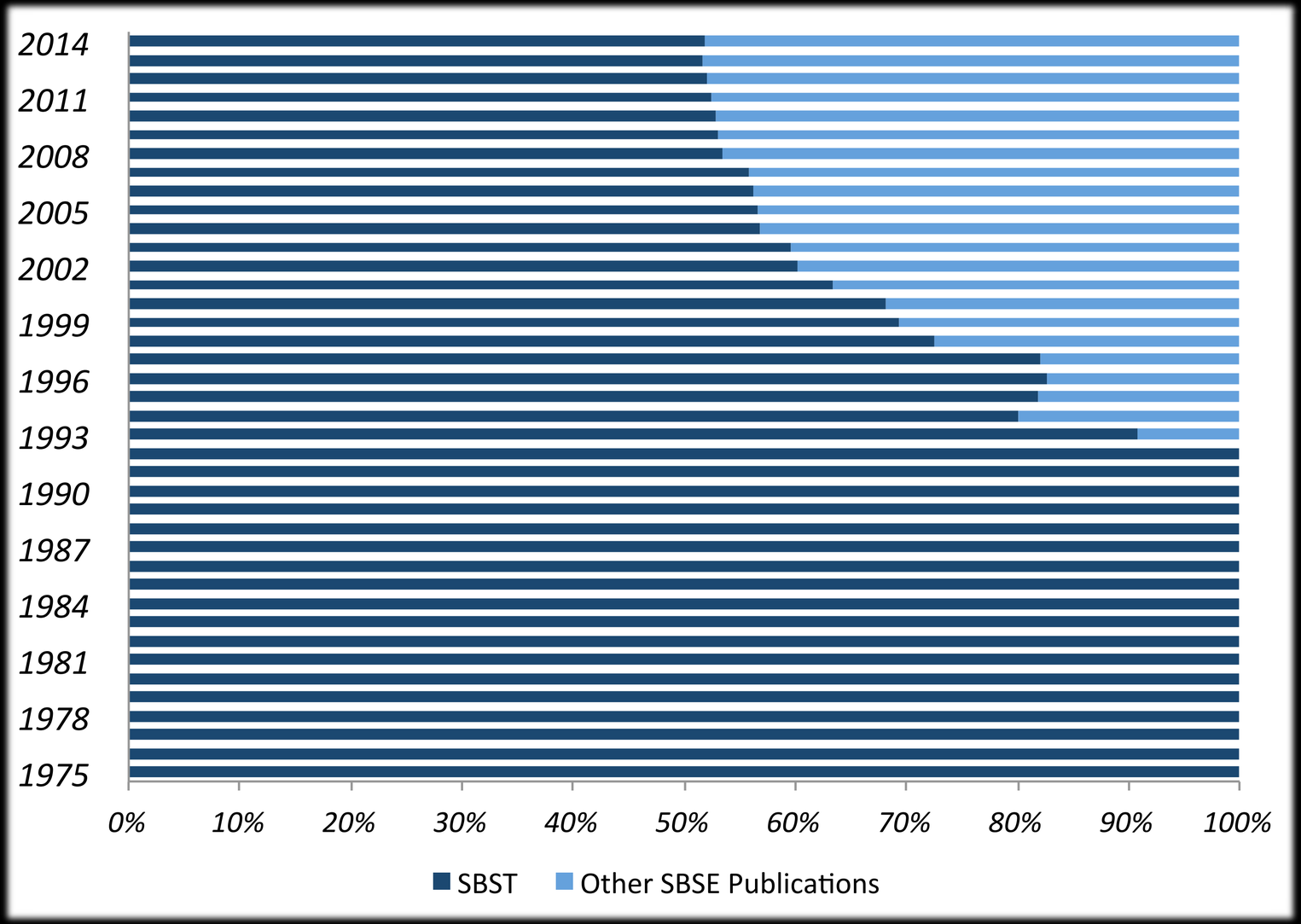
Careful human-based update

100% precision and recall





Polynomial yearly rise in the number of papers Search Based Software Testing



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.

SBST

History

Structural

Structural

find tests to cover branches, statements & dataflow, etc.

Integration

Integration

find
best component
ordering

Temporal

Temporal

find worst case execution time

CIT

CIT

find 2-way, 3-way
n-way
interaction tests

SPLs

Augment

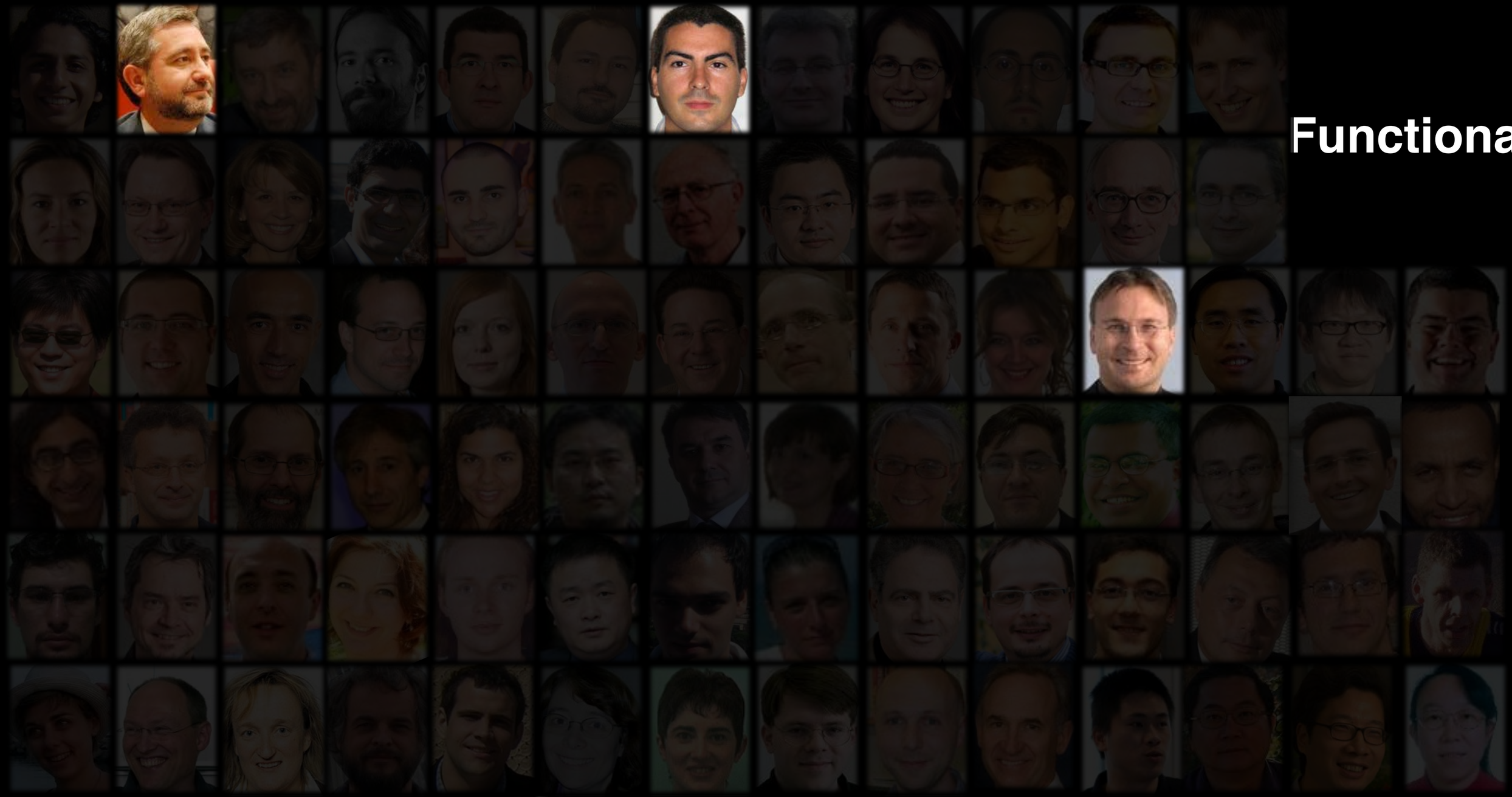
Augment

find new tests from old tests

Regression

Regression

find good subsets and orders of tests



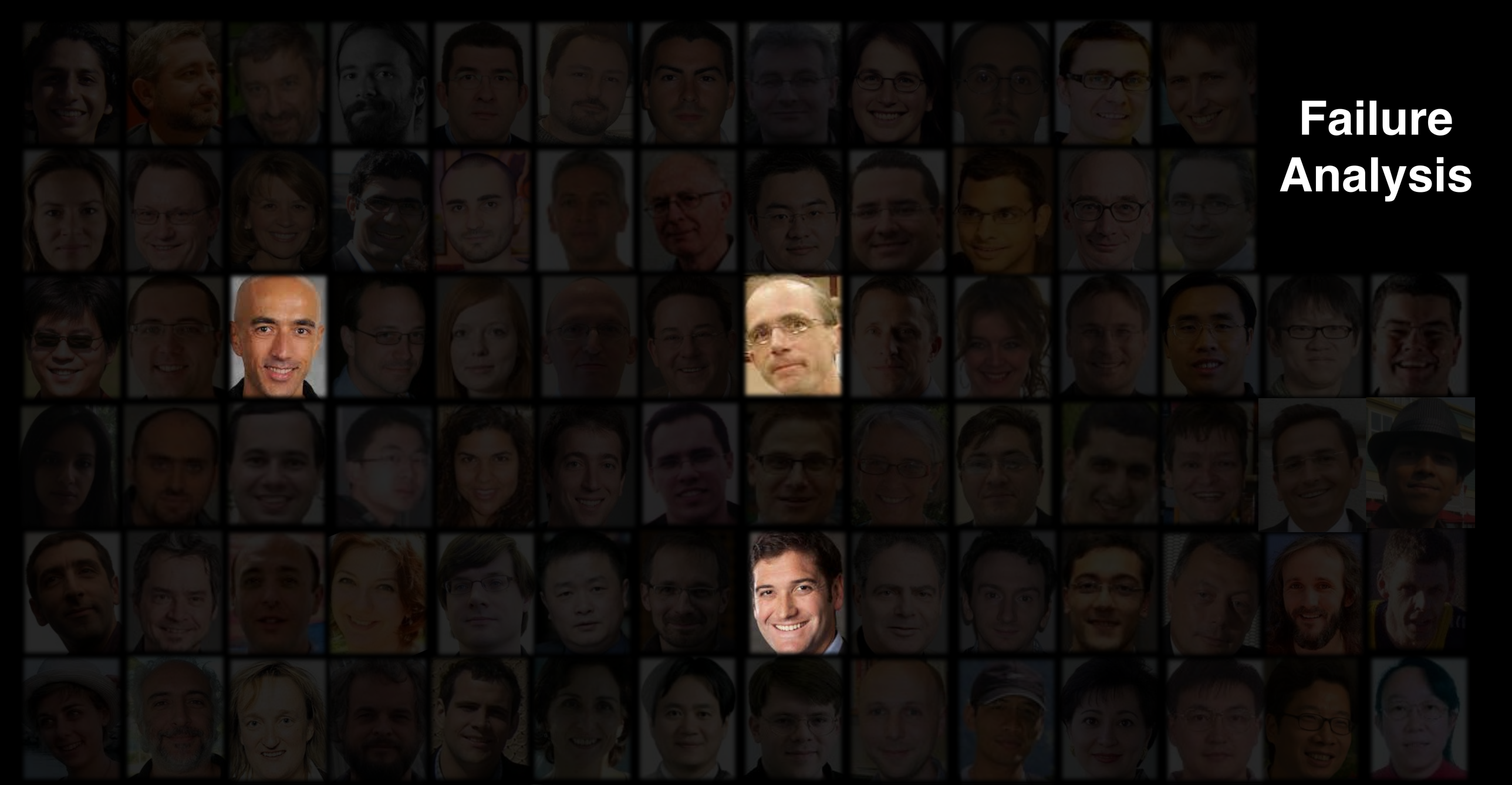
Functional

Mutation

State based

Model based

Black box



Failure Analysis

Security

Web/ Services

Agents

SBST's Industrial Applications and Tools

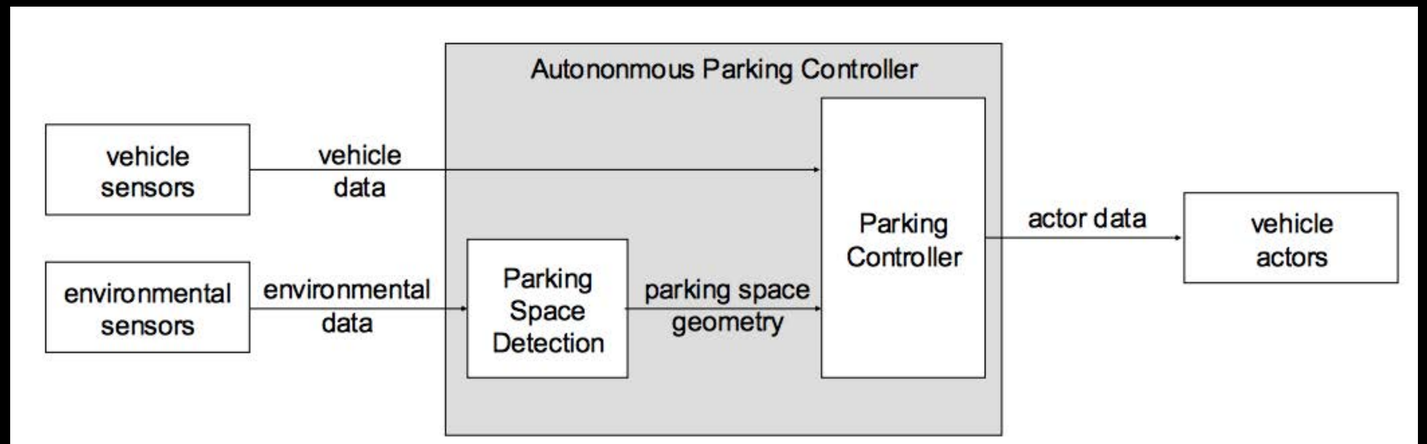
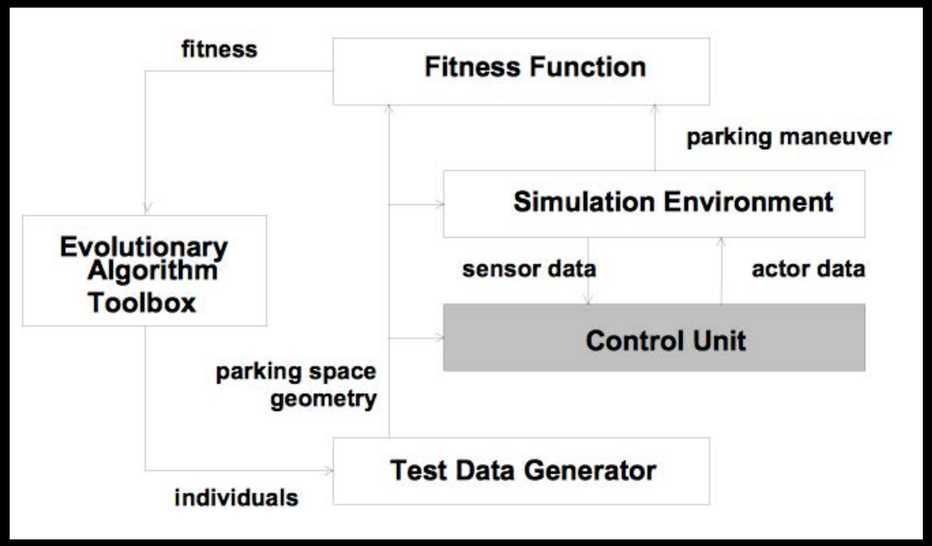


Fig.2: System Environment and Sub-Components of the Autonomous Parking System



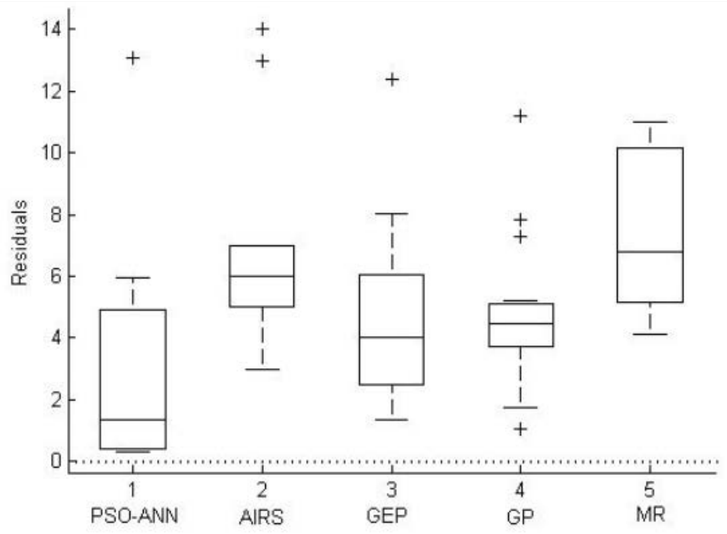
Joachim Wegener and Oliver Bühler. GECCO 2004

DAIMLER

SBST's Industrial Applications and Tools

Table I
VARIABLES OF INTEREST FOR THE PREDICTION MODELS.

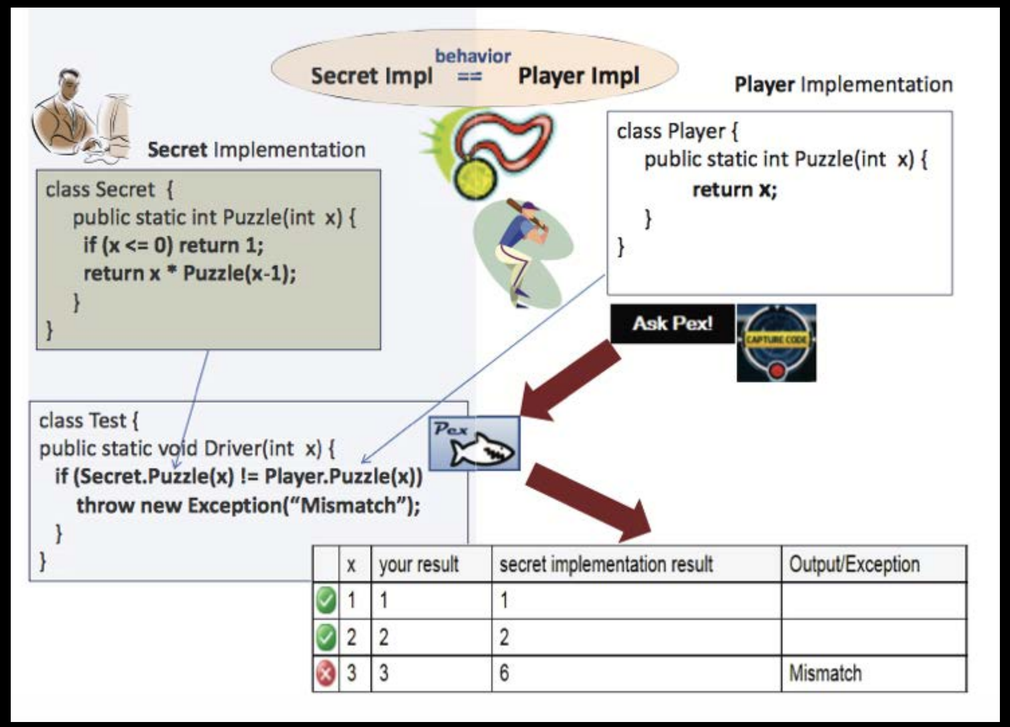
No.	*Description*	*Abbreviation*	*Category*
1	Fault in-flow	F. in-flow	Fault-inflow
2	No. of work packages planned for system integration	No. WP. PL. SI	Status rankings of WPs
3	No. of work packages delivered to system integration	No. WP. DEL. SI	
4	No. of work packages tested by system integration	No.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



(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

SBST's Industrial Applications and Tools



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

Microsoft

SBST Public Tools

austin-sbst

AUgmented Search-based TestINg

Project Home | Downloads | Wiki | Issues | Source

Summary | People

Project Information

Starred by 2 users
Project feeds

Code license
New BSD License

Labels
C, Ocaml, Academic, Research, Tool

Members
kiran.la...@gmail.com

AUSTIN is a structural test data generation tool (for unit tests) for the C language. I
project is to aid researchers in automated test data generation using search-based
supports a random search, as well as a simple hill climber that is augmented with a

If you've downloaded AUSTIN, please let me know. If you are using AUSTIN as par
then please use the following bibtex entry.

@inproceedings {austin-lakhotia,

Author = {Kiran Lakhotia and Mark Harman and Hamilton Gross},

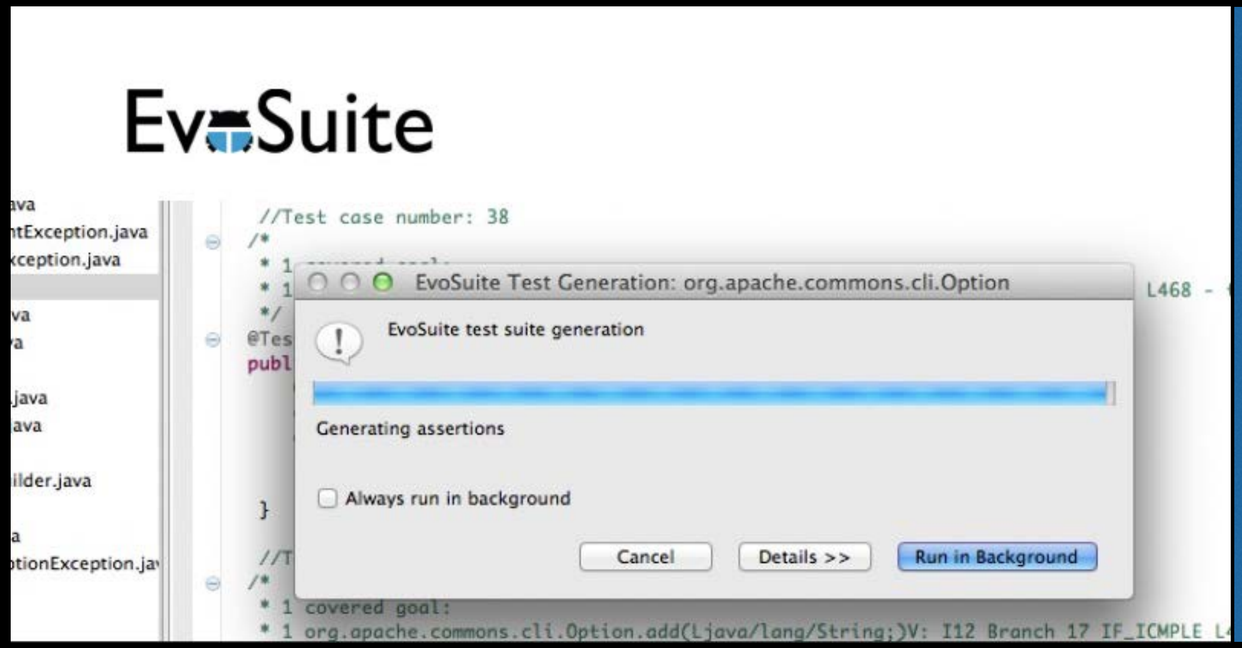
Booktitle = {2nd International Symposium on Search Based Software Engineering},

Pages = {101--110},

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 high recommended tool.

Gordon Fraser and Andrea Arcuri. ESEC/FSE 2011

More details in the keynote paper in your ICST proceedings

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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

II. A Brief History of SBST

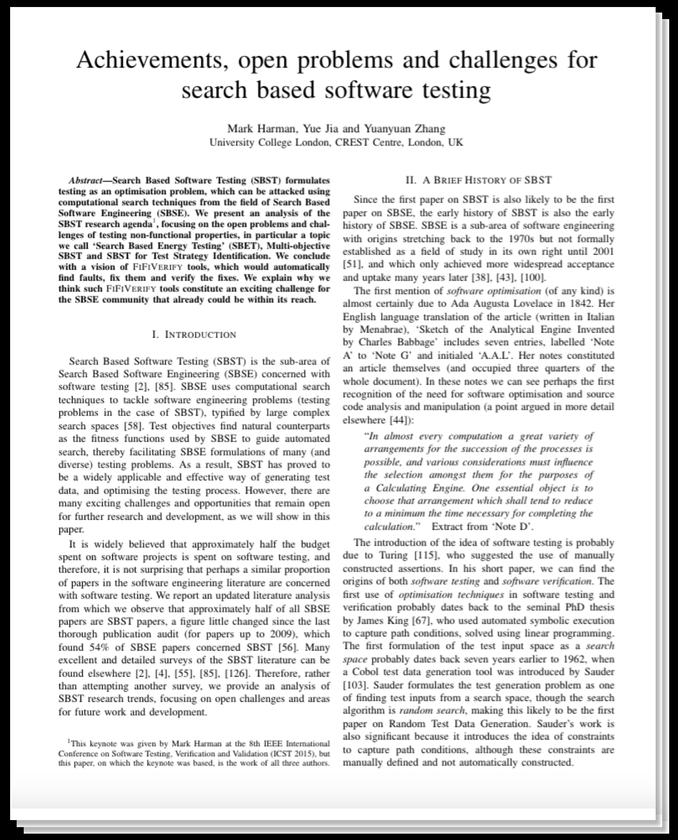
Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 [51], 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 Menabrae), '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]):

> "*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'.

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 automated 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.

SBST's Challenges



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

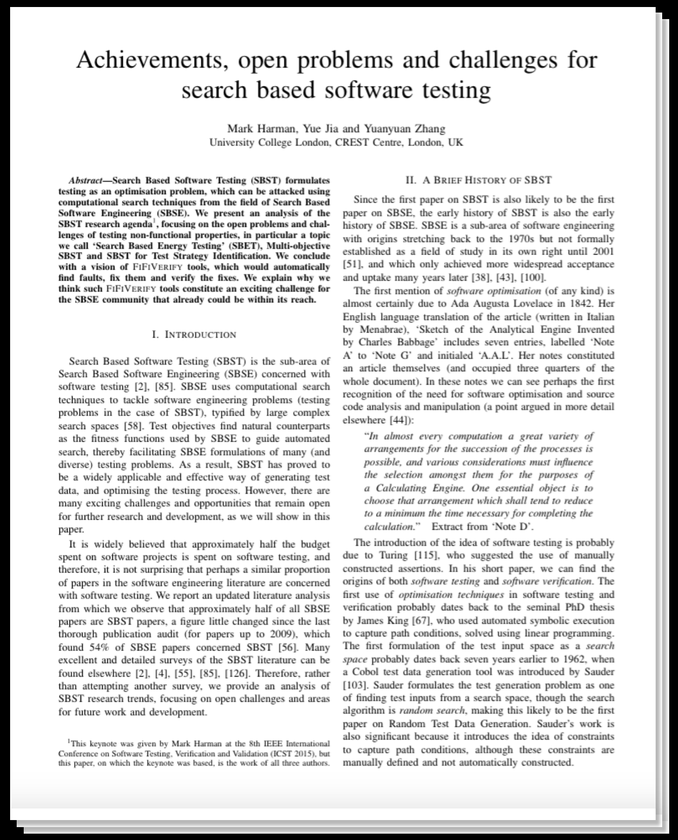
The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both *software testing* and *software verification*. The first use of *optimisation techniques* in software testing and verification probably dates back to the seminal PhD thesis by James King [67], who used automated 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.

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



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

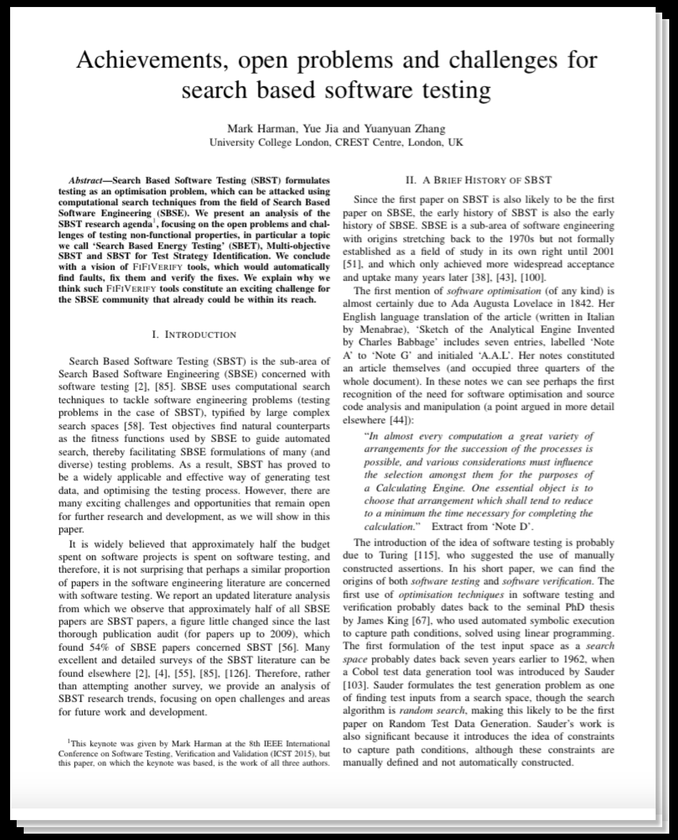
The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both *software testing* and *software verification*. The first use of *optimisation techniques* in software testing and verification probably dates back to the seminal PhD thesis by James King [67], who used automated 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.

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



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

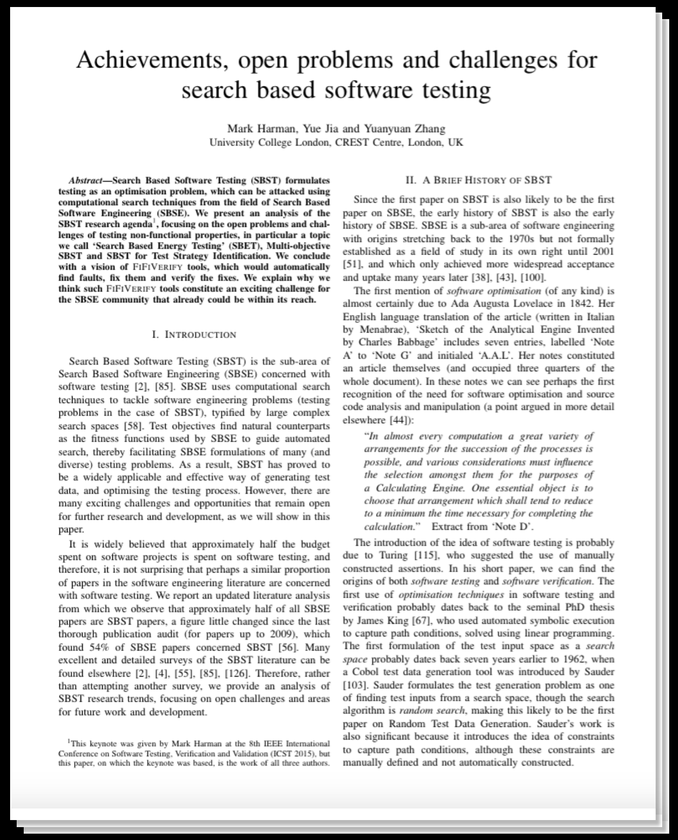
The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both *software testing* and *software verification*. The first use of *optimisation techniques* in software testing and verification probably dates back to the seminal PhD thesis by James King [67], who used automated 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.

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



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

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 automated 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.

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).

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

Analysis

SBST for Non-Functional Properties

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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

II. A Brief History of SBST

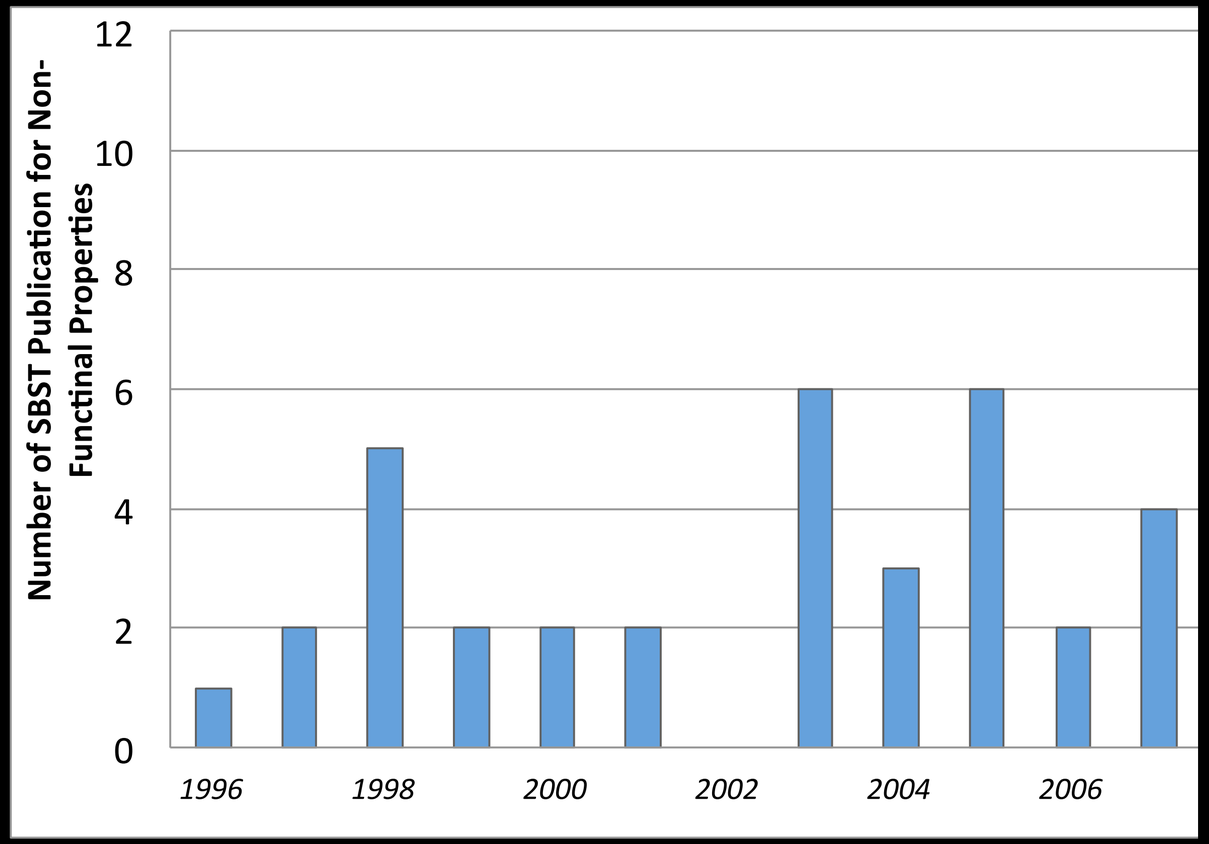
Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 [51], 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 Menabrae), '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]):

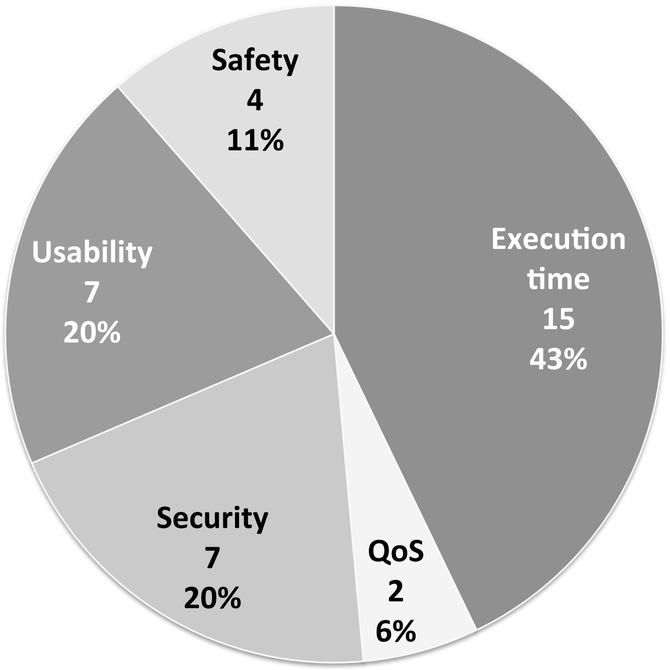
> "*In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce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

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 automated 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.

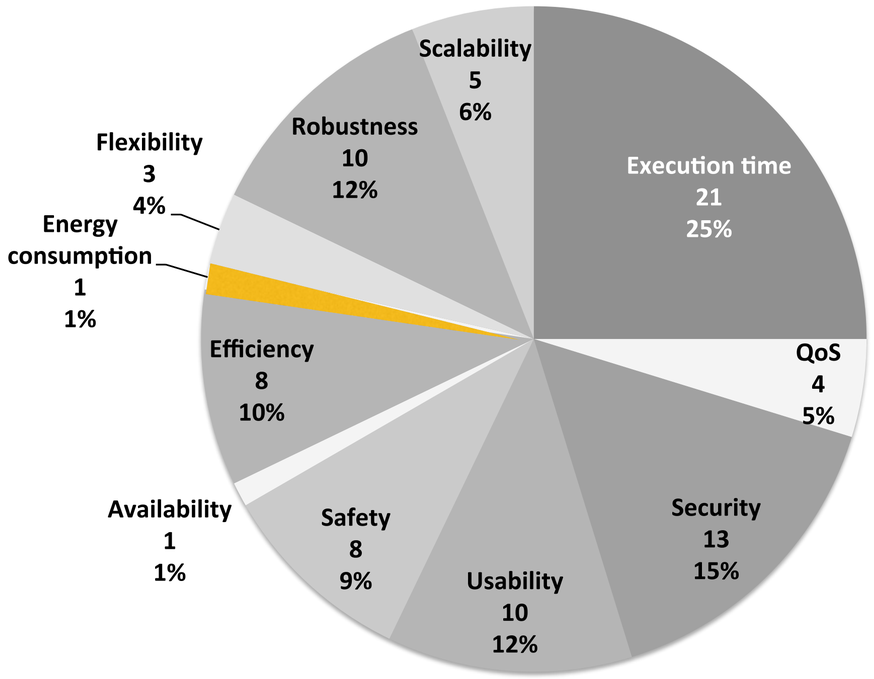
SBST for Non-Functional Properties



Mark Harman, Wasif Afzal, Yuanyuan Zhang, Tao Xie, Jian Zhang, ICST 2015 Field Extensions 2009 et al 2009.



The Categories of Non-Functional Properties from 1996 to 2007



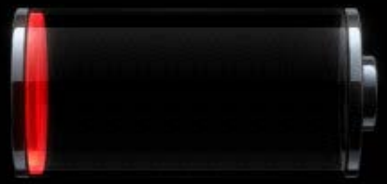
The Categories of Non-Functional Properties from 1996 to 2014

Changing contributions of SBET

Tiny amount on SBST 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)

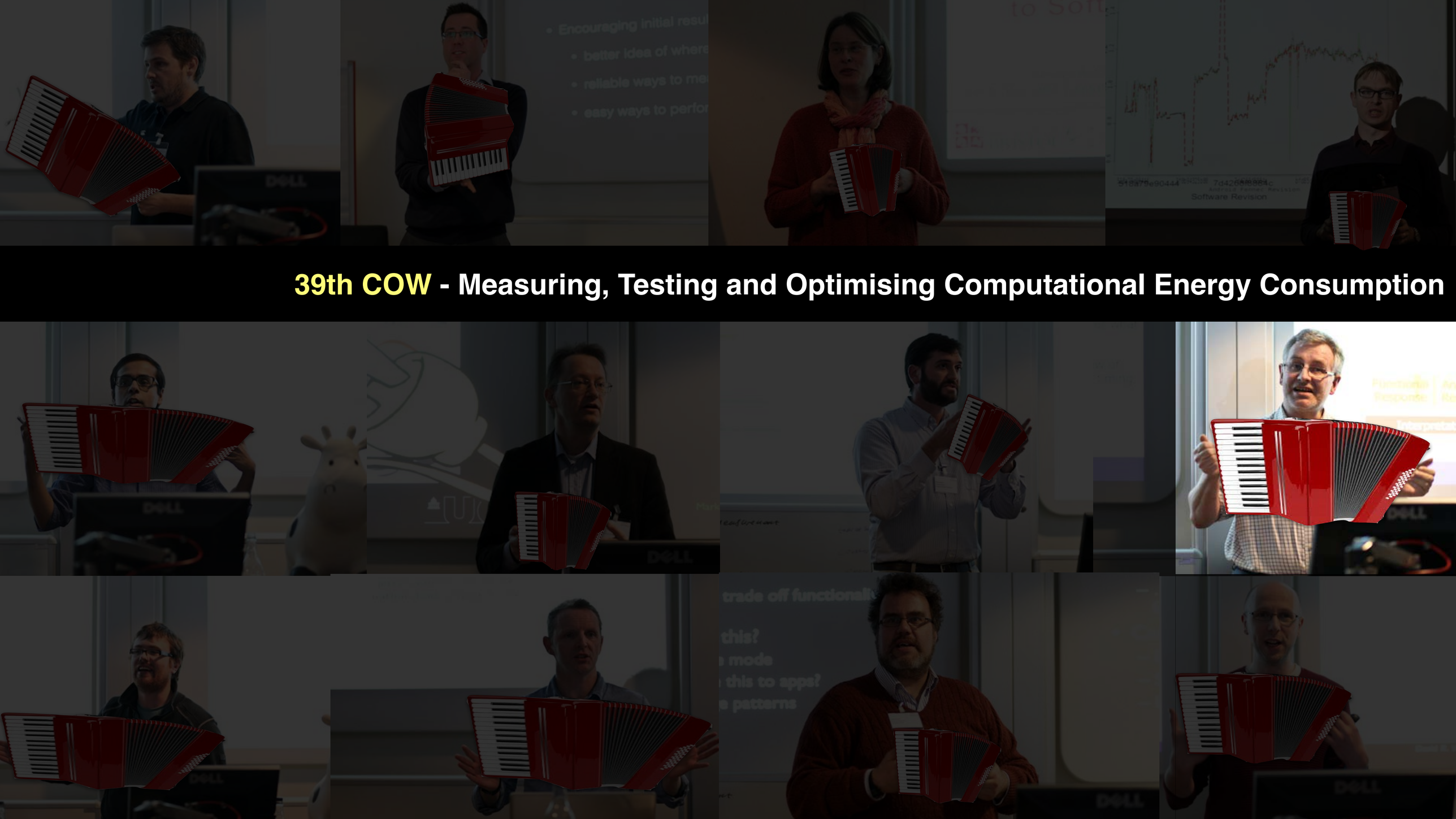
Efficiency: will need to consider many different test cases

Coarse Granularity: Energy consumed per run overall

Hawthorne Effect: Instrumentation may affect energy consumed

39th COW - Measuring, Testing and Optimising Computational Energy Consumption

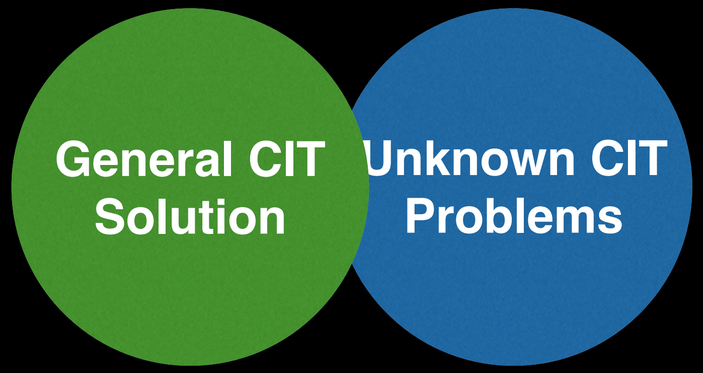
39th COW - Measuring, Testing and Optimising Computational Energy Consumption



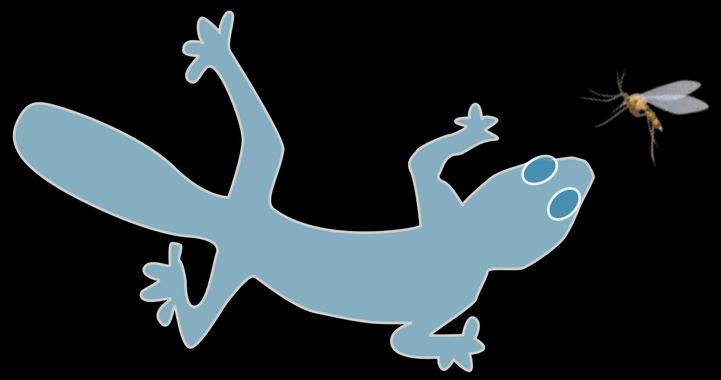
39th COW - Measuring, Testing and Optimising Computational Energy Consumption

Search Based Test Strategy Identification (SBTSI)

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



A hyperheuristic SBSTI for CIT



A co-evolutionary SBSTI for Mutation testing

A hyperheuristic SBSTI for CIT

CIT Solutions

AETG

IPOG

GA

Simulated Annealing

Tabu

Hill Climbing

SBSTI

A hyperheuristic SBSTI for CIT

CIT Solutions

AETG

IPOG

GA

Simulated Annealing

Tabu

Hill Climbing

CIT Problem Characteristics

Specific structures

unconstrainted 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

unconstrainted 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

unconstrainted 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

uncon roblems

Unknown CIT Problems

constraint oblems

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

constraint problems

Weight

SBSTI

A hyperheuristic SBSTI for CIT

CIT Solutions

AETG

IPOG

General CIT Solution

Tabu

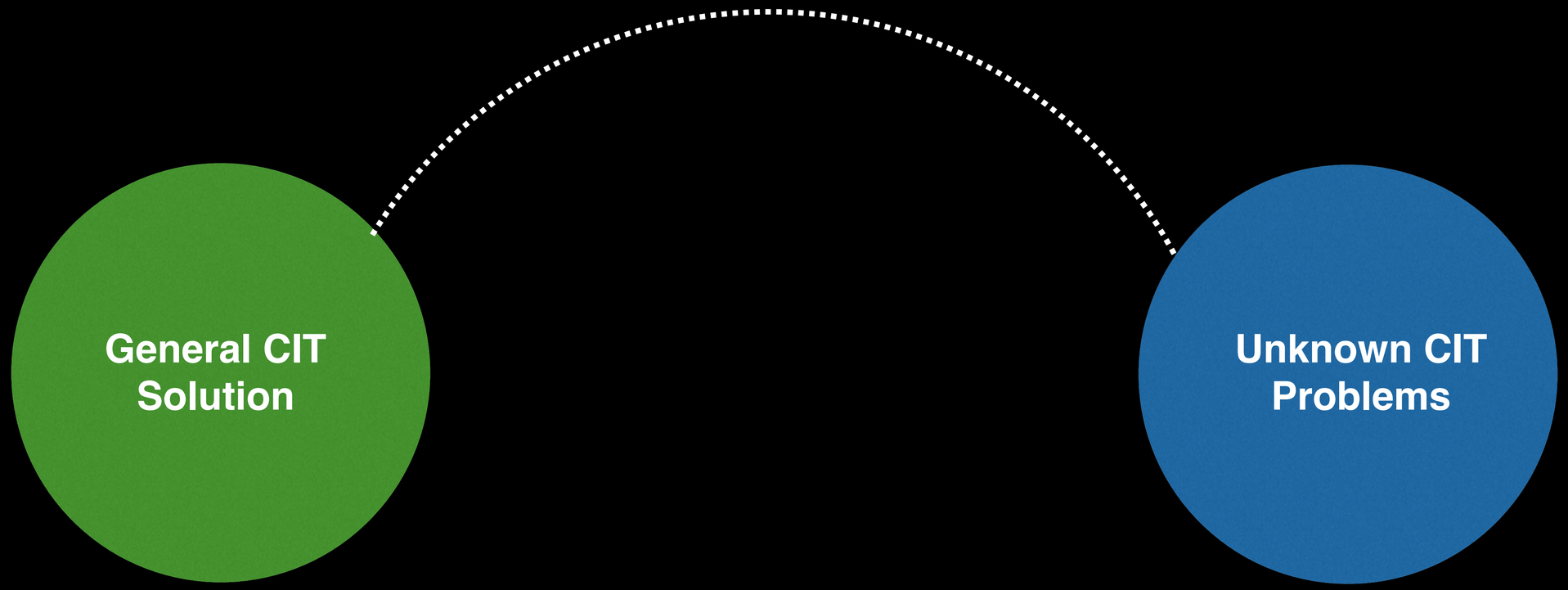
Hill Climbing

CIT Problem Characteristics

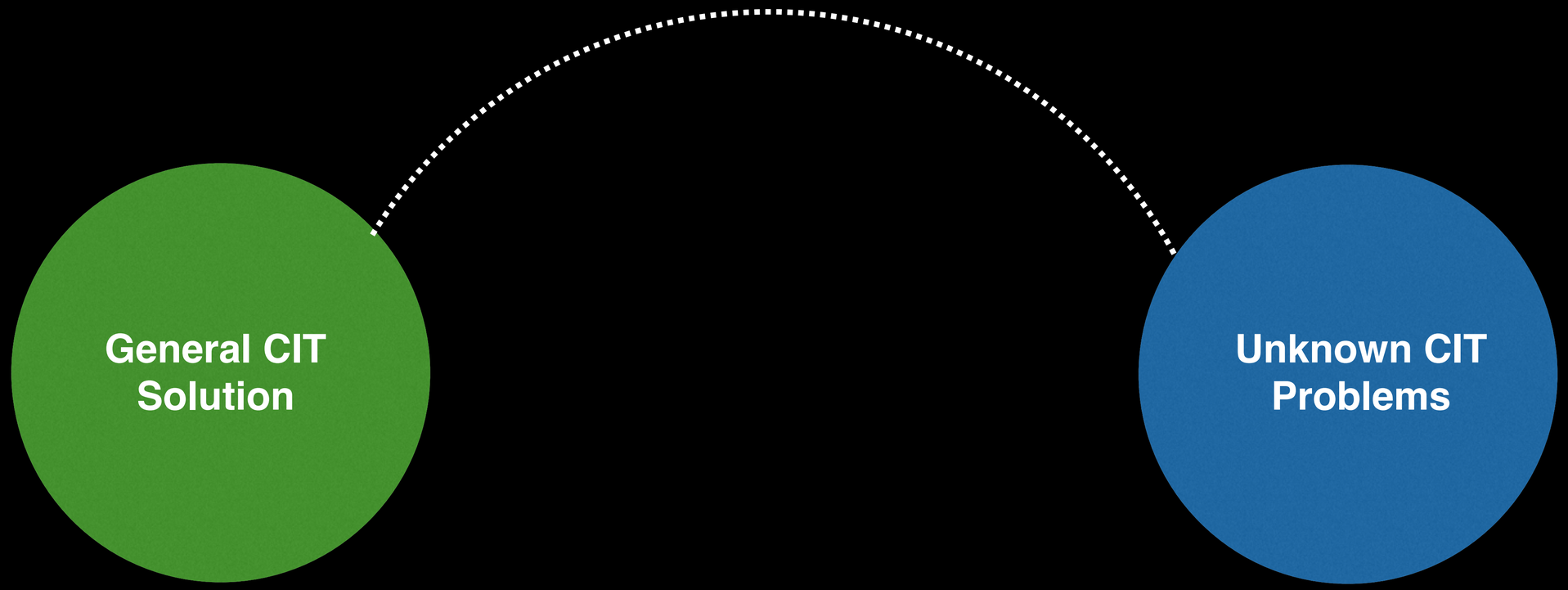
Specific structures

Unknown CIT Problems

SBSTI



Learning Combinatorial Interaction Test Generation Strategies using Hyperheuristic Search. Yue Jia, Myra Cohen, Mark Harman and Justyna Petke. ICSE 2015



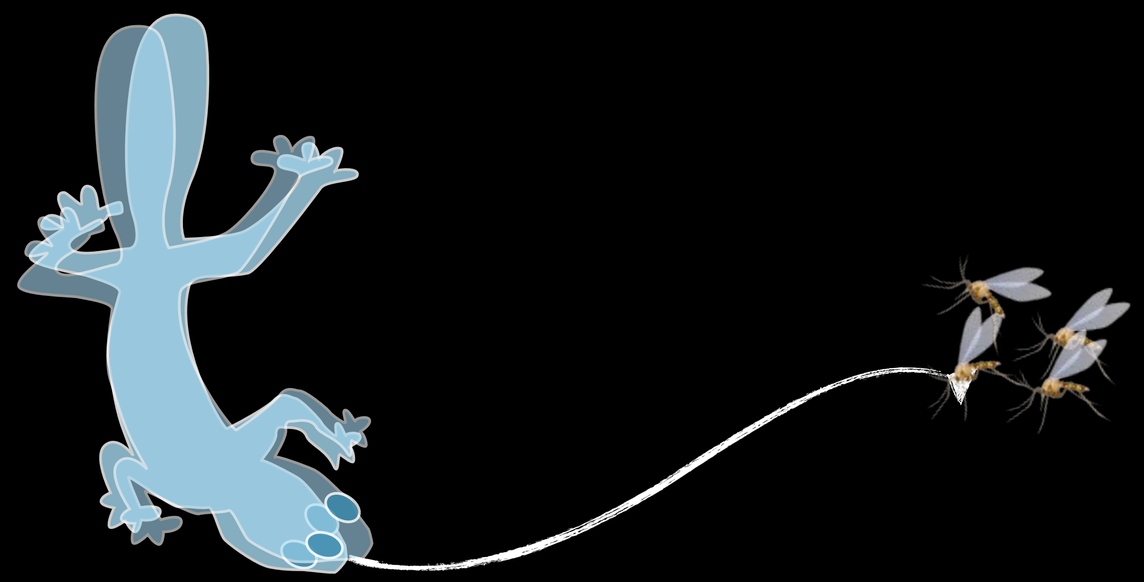
Learning Combinatorial Interaction Test Generation Strategies using Hyperheuristic Search. Yue Jia, Myra Cohen, Mark Harman and Justyna Petke. ICSE 2015

A co-evolutionary approach for SBSTI

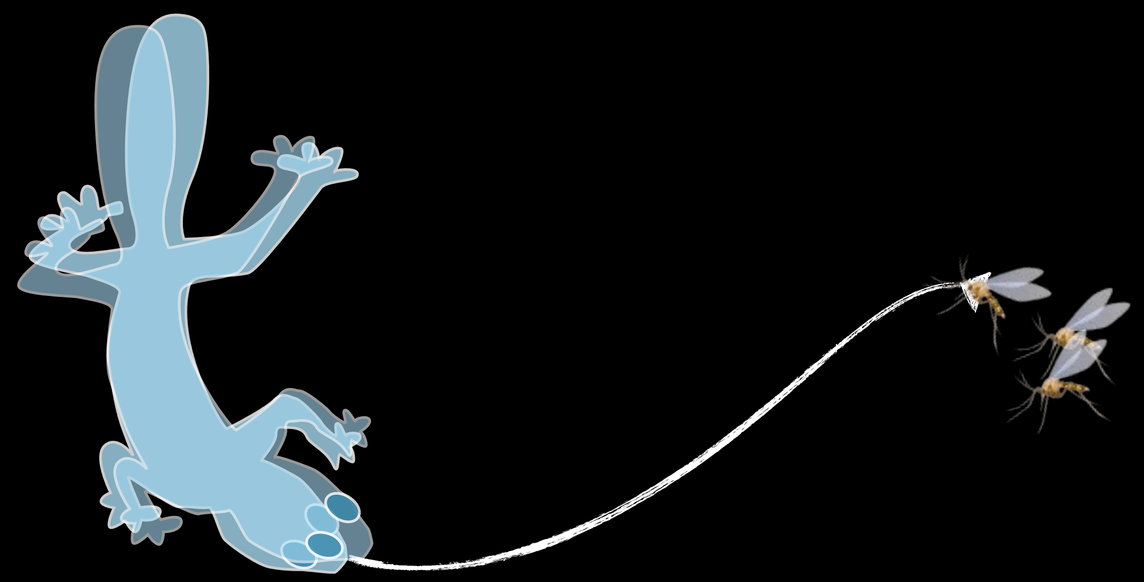
Predator	Prey
Testing	Bugs

A co-evolutionary approach for SBSTI

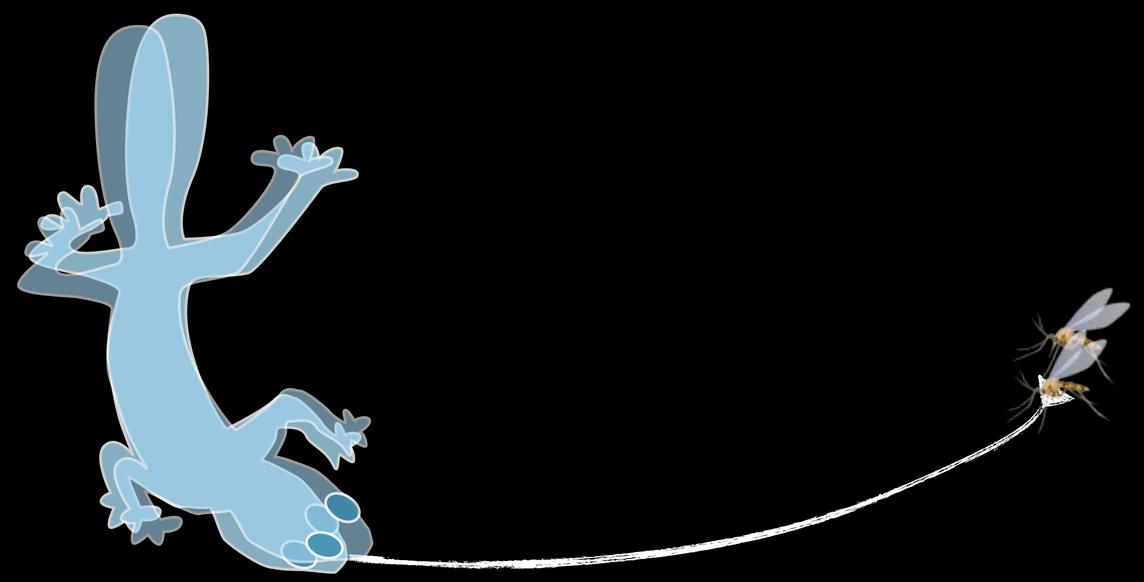
A co-evolutionary approach for SBSTI



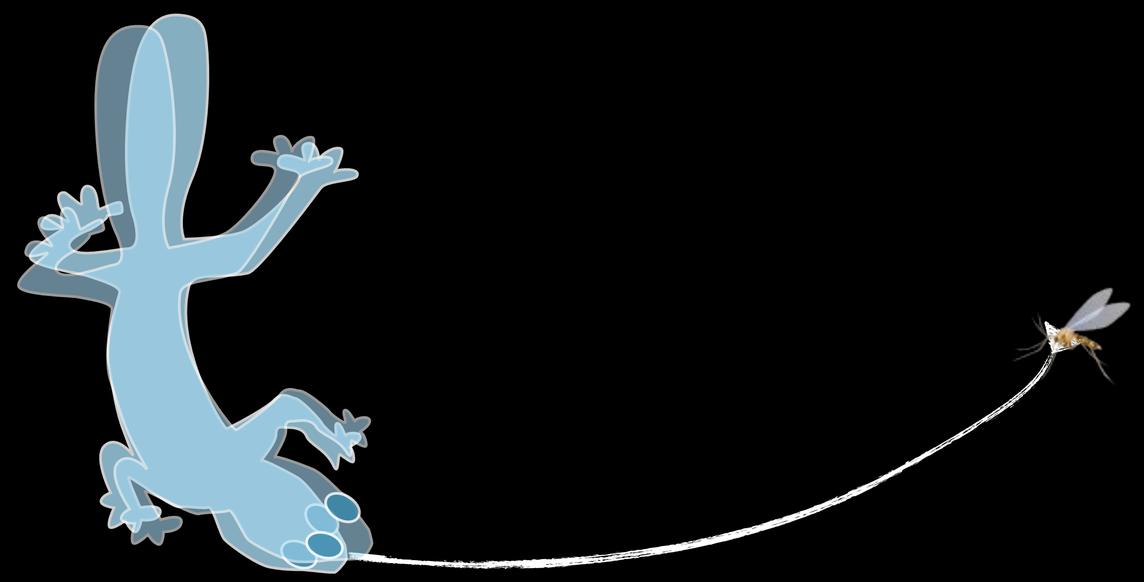
A co-evolutionary approach for SBSTI



A co-evolutionary approach for SBSTI



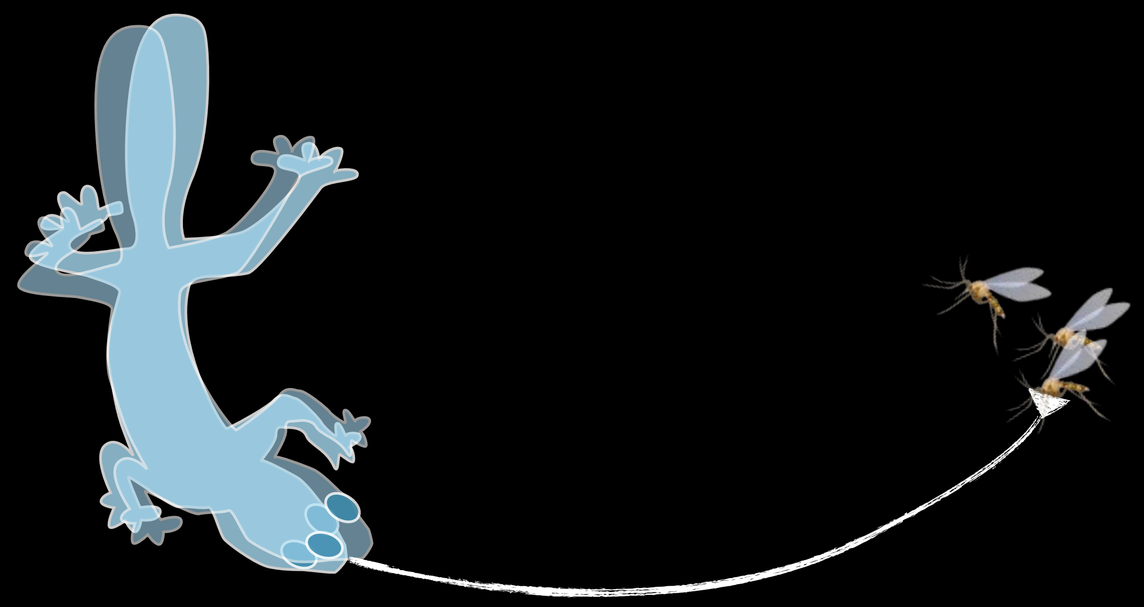
A co-evolutionary approach for SBSTI



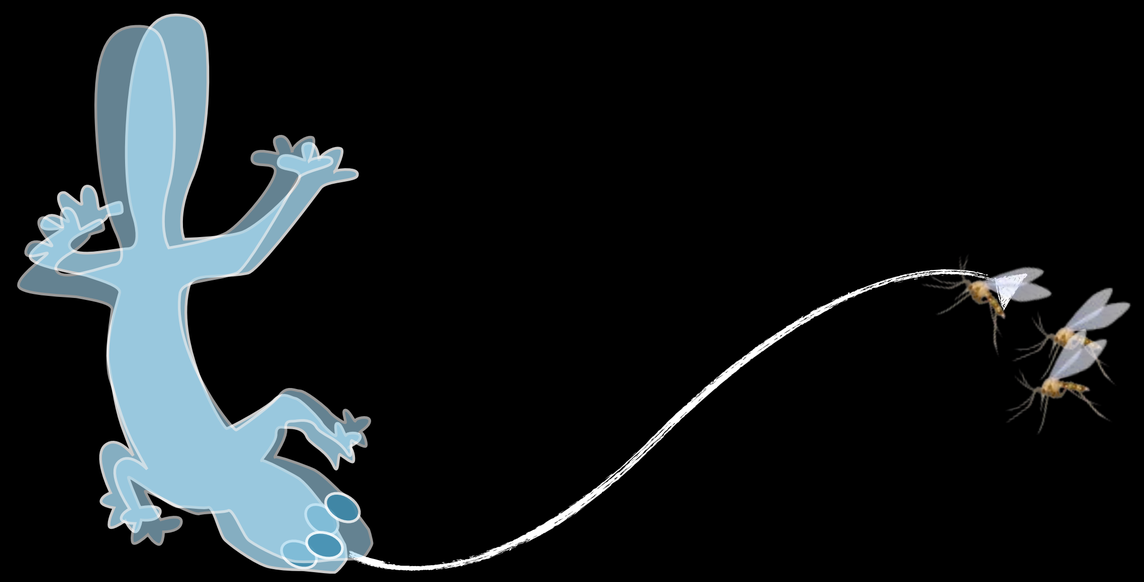
A co-evolutionary approach for SBSTI

Evolving

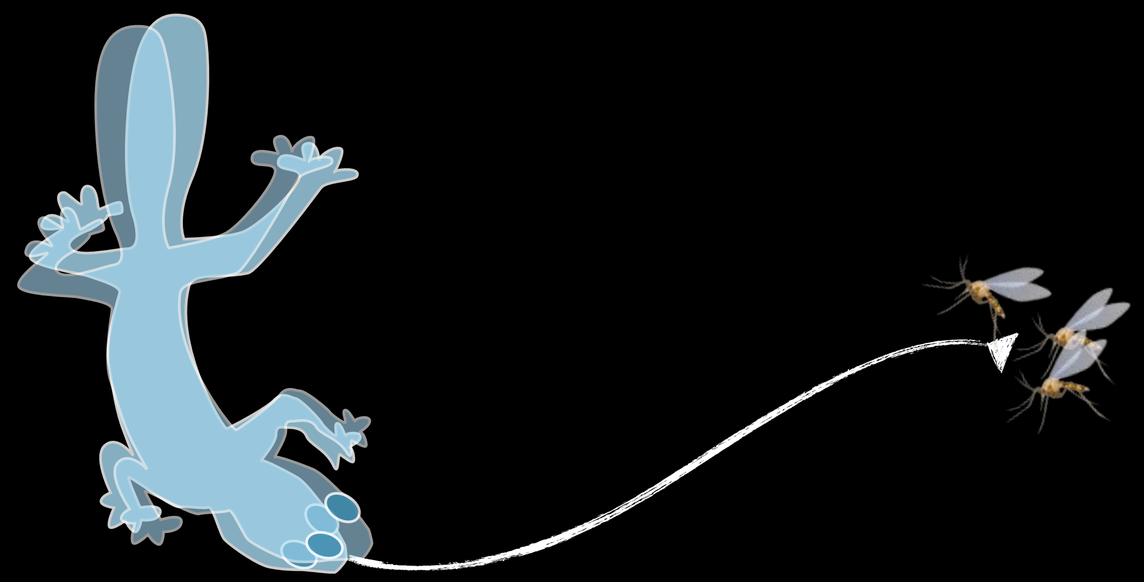
A co-evolutionary approach for SBSTI



A co-evolutionary approach for SBSTI



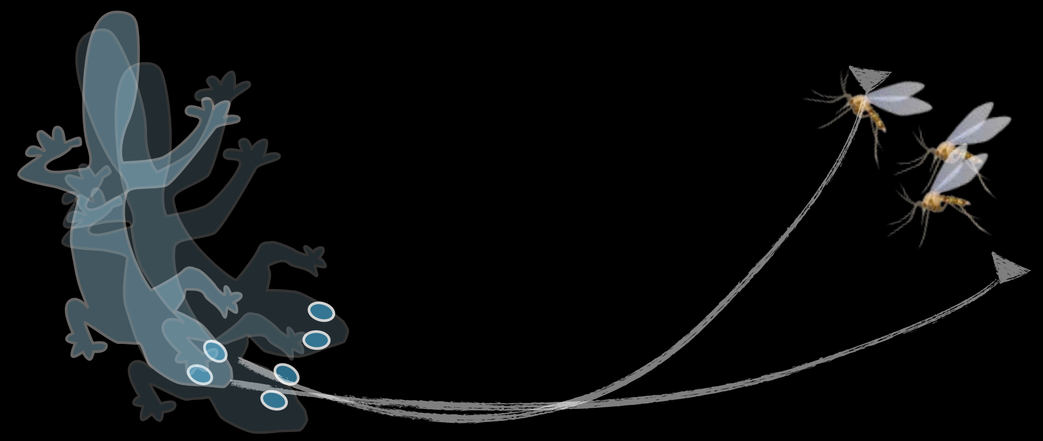
A co-evolutionary approach for SBSTI



A co-evolutionary approach for SBSTI

Evolving

A co-evolutionary approach for SBSTI



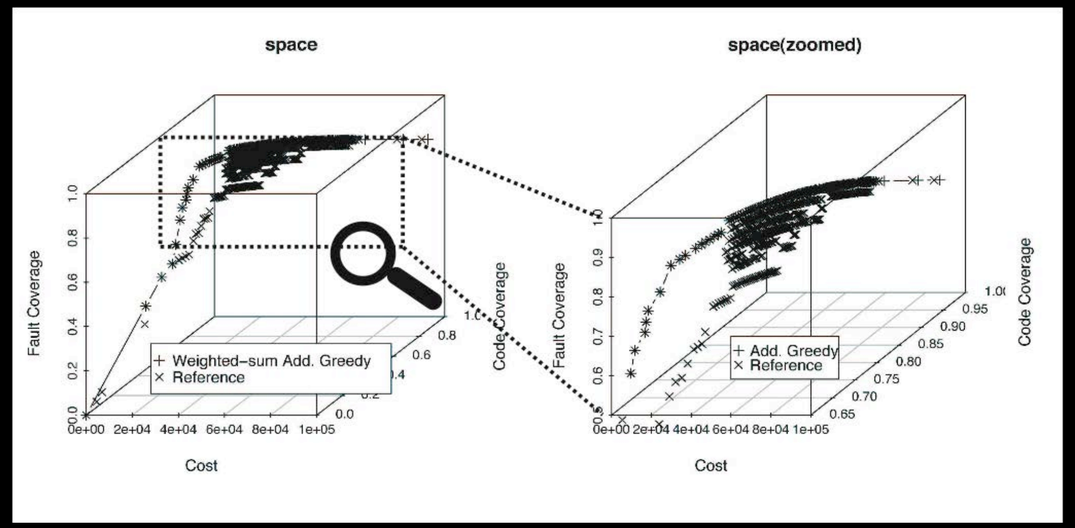
A co-evolutionary approach for SBSTI

Test data

Higher order mutants

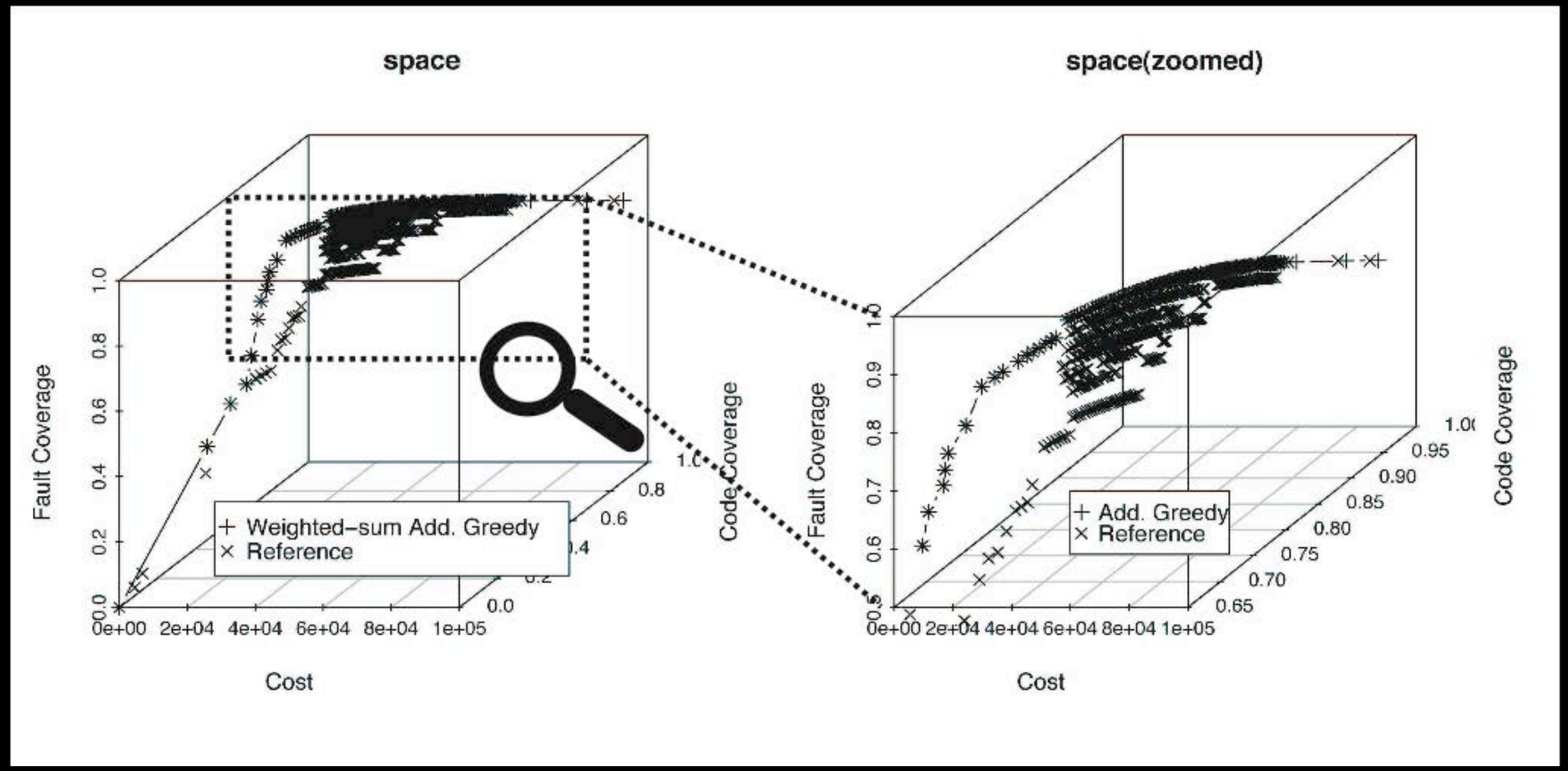
Multi-Objective Search Based Testing (MOSBAT)

increasingly prevalent
regression testing was early adopter



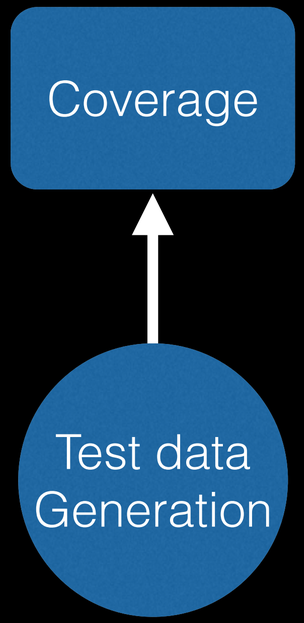
e.g. Yoo and Harman: ISSTA 2007

Multi-Objective Search Based Testing (MOSBAT)

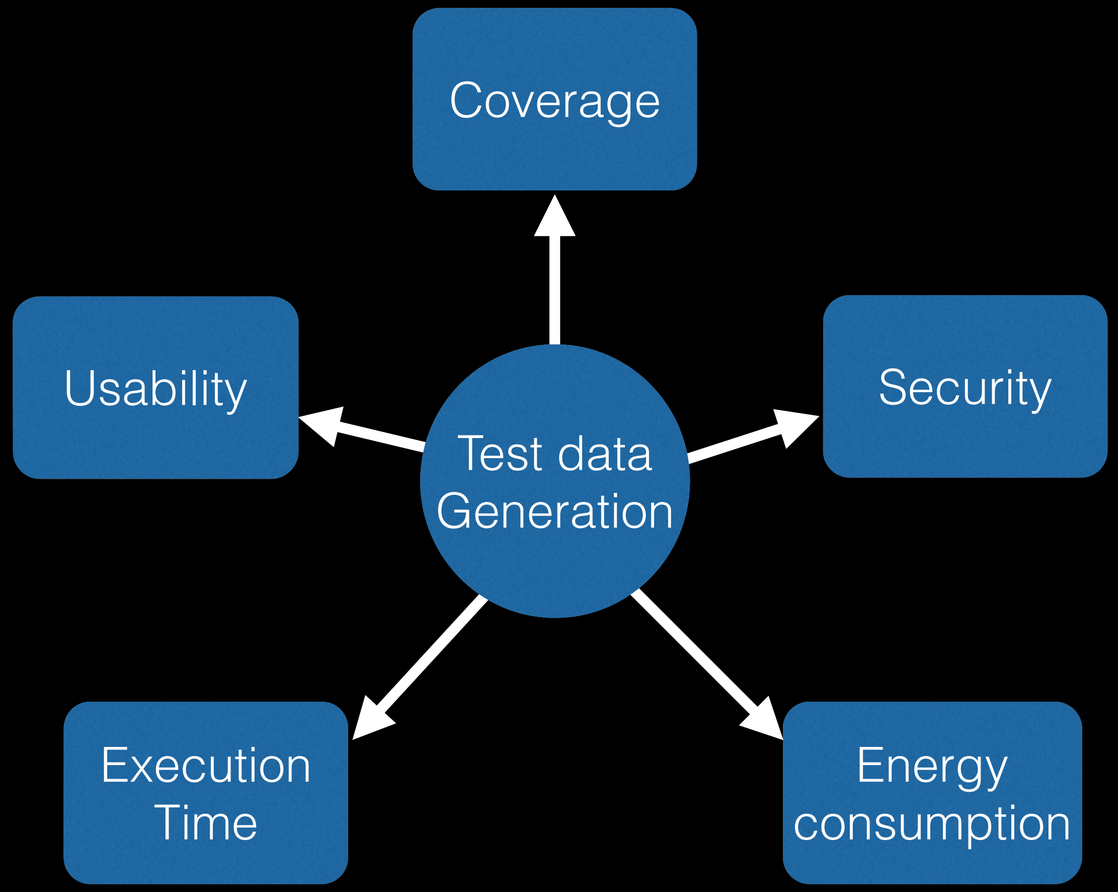


Multi-Objective Search Based Testing (MOSBAT)

test case generation
is still mostly single objective

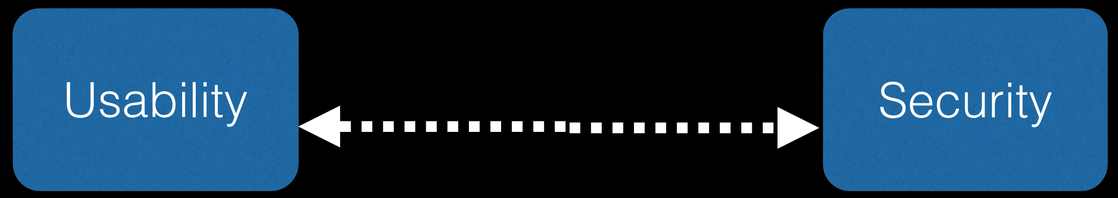


Multi-Objective Search Based Testing (MOSBAT)

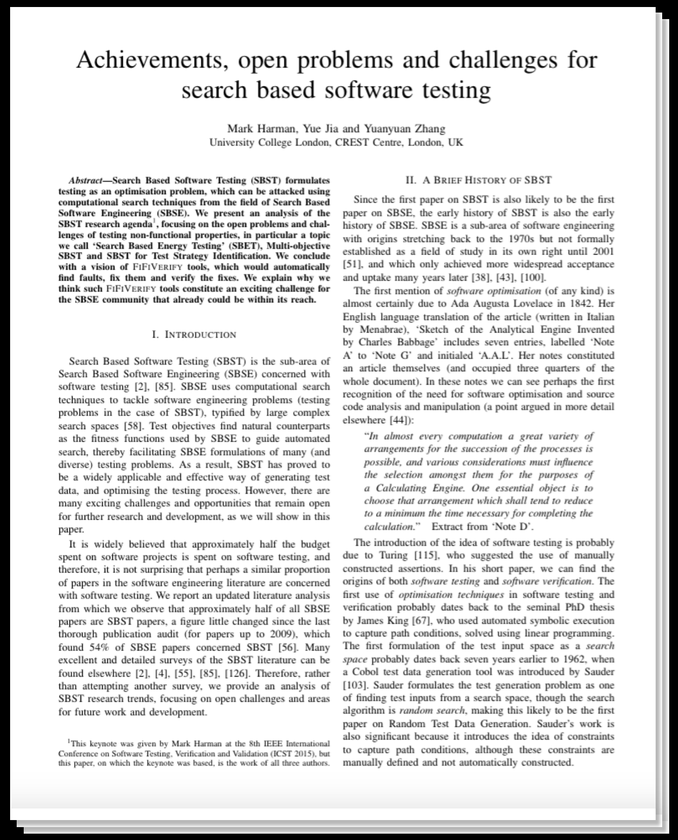


Multi-Objective Search Based Testing (MOSBAT)

Multi-objective Understanding:
Debug Security policies



SBST's Challenges



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> *"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'.

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 automated 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.

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

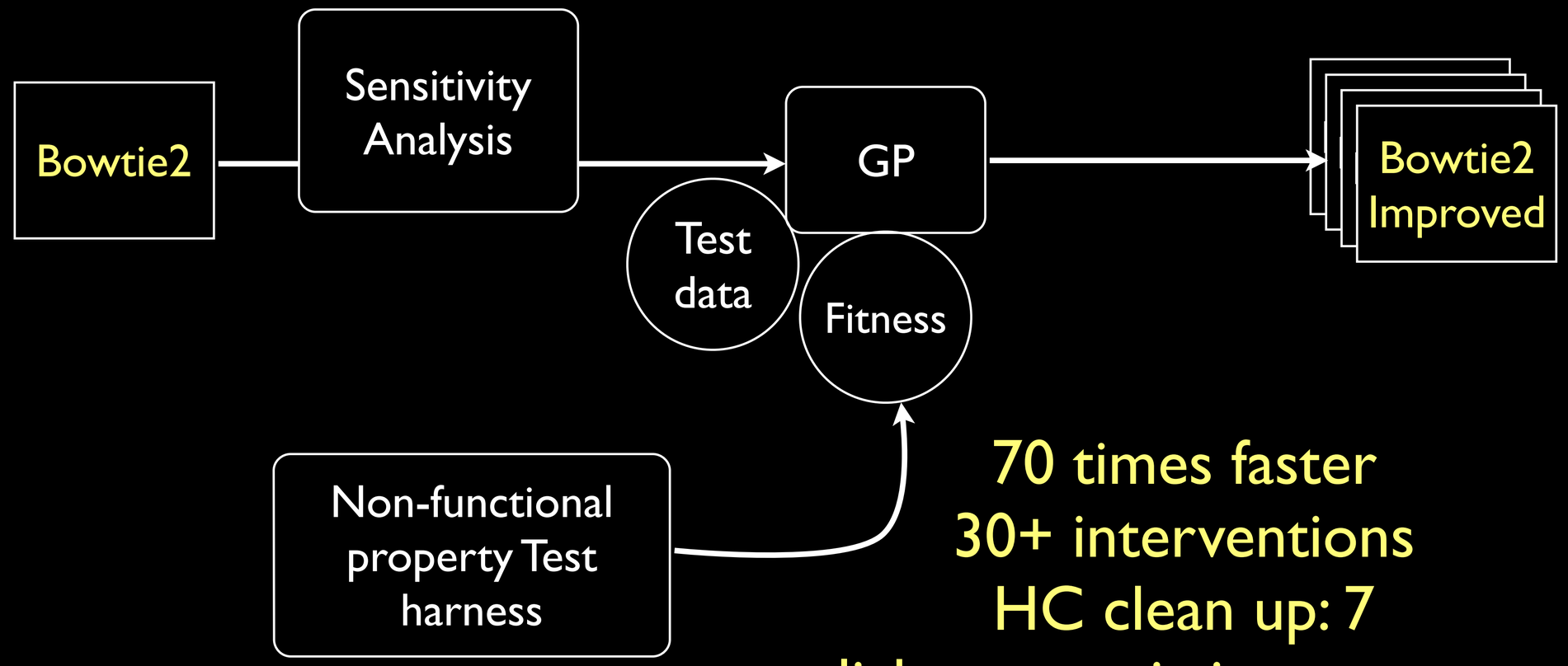
We need Search Based Test Strategy Identification (SBTSI).

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

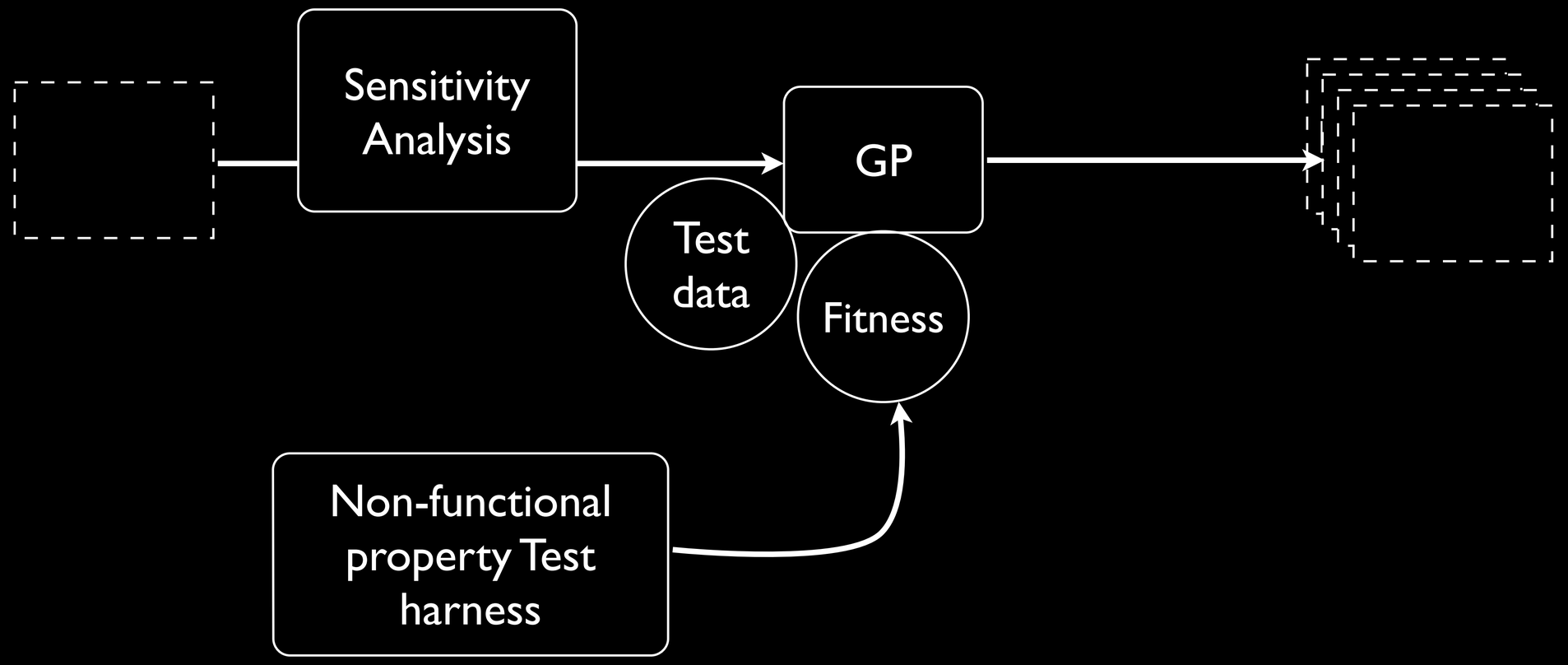
Genetic Improvement of Programs



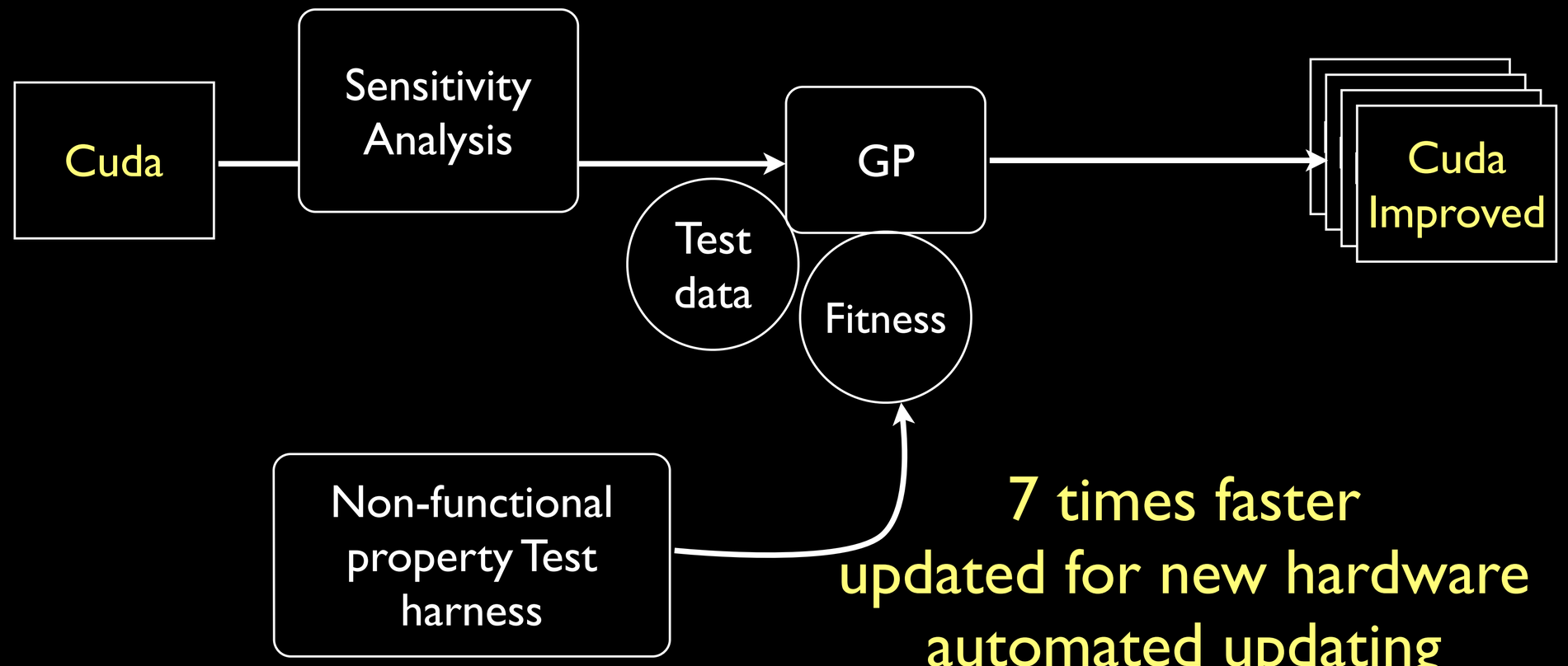
70 times faster
30+ interventions
HC clean up: 7
slight semantic improvement

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

Genetic Improvement of Programs



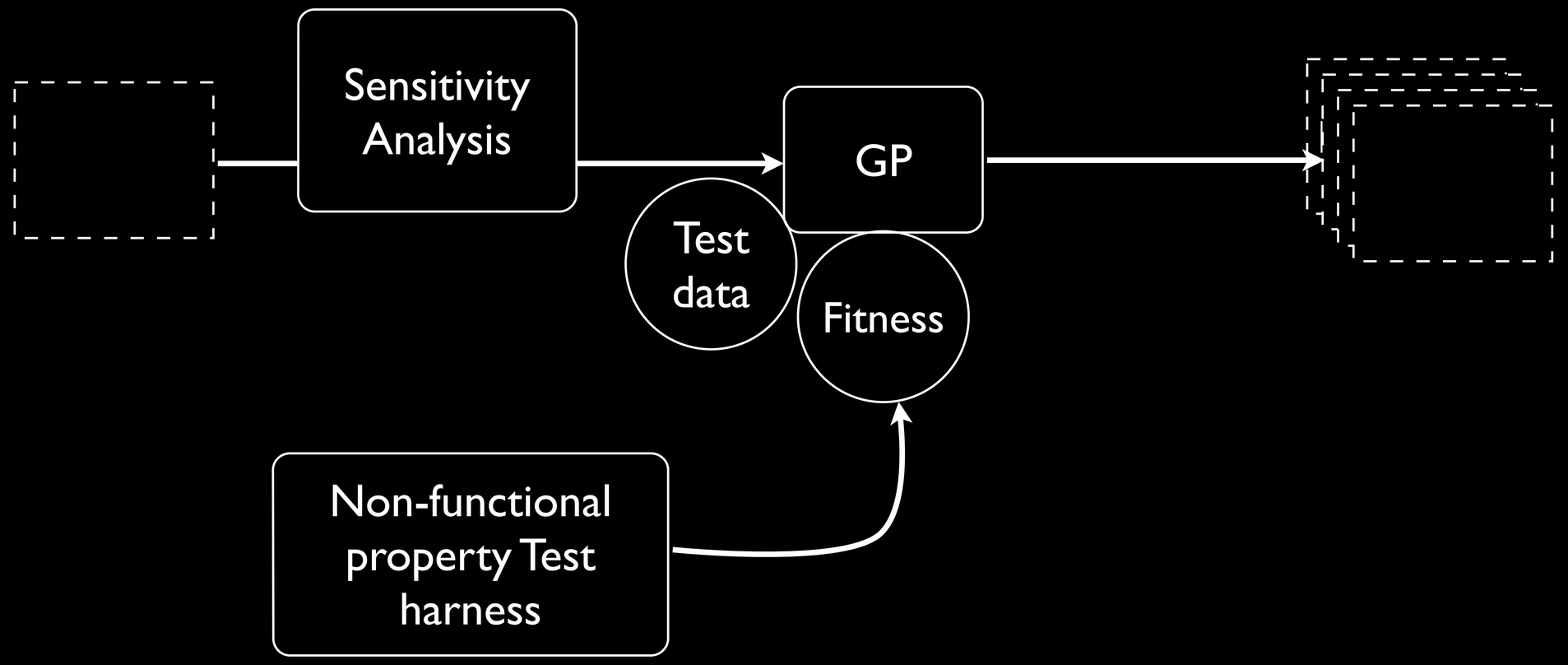
Genetic Improvement of Programs



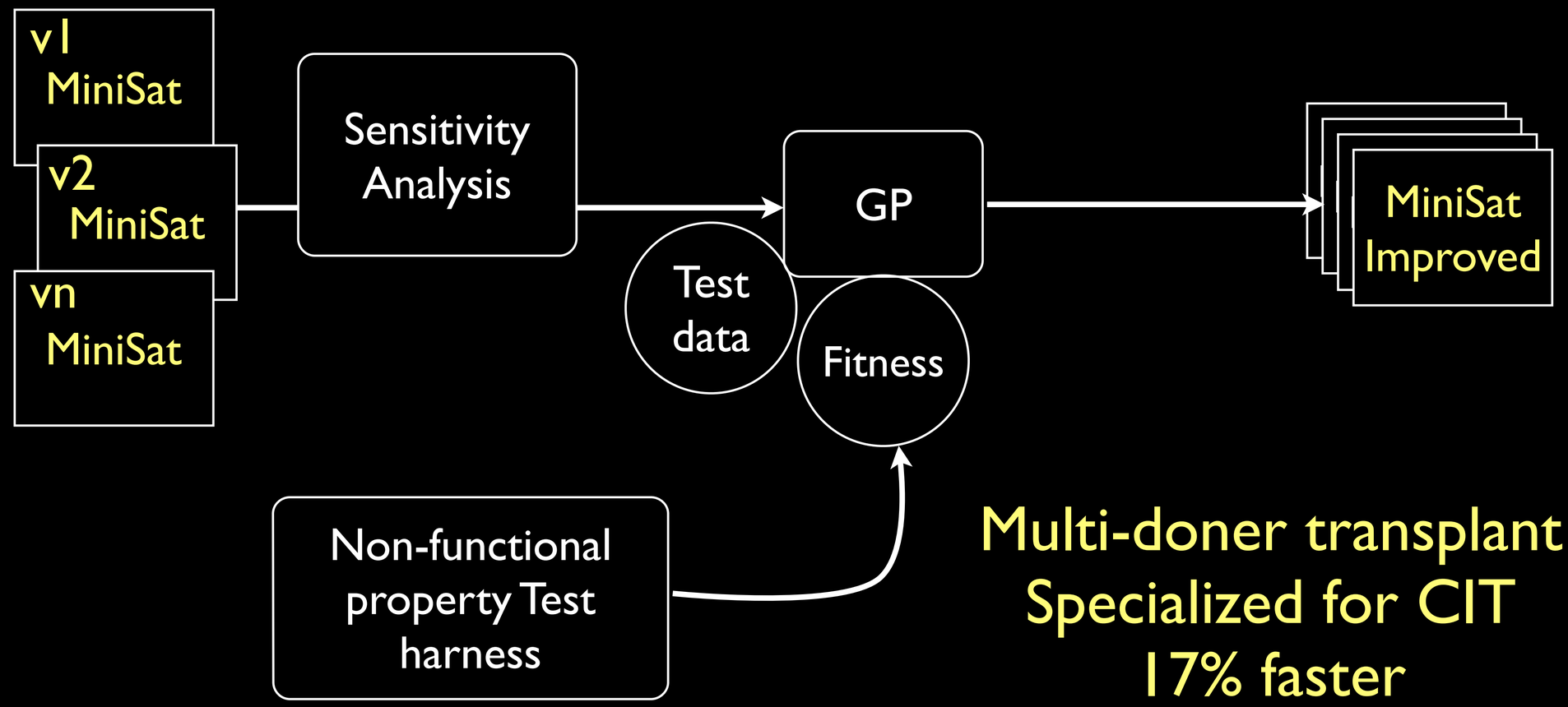
7 times faster
updated for new hardware
automated updating

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

Inter version transplantation



Inter version transplantation

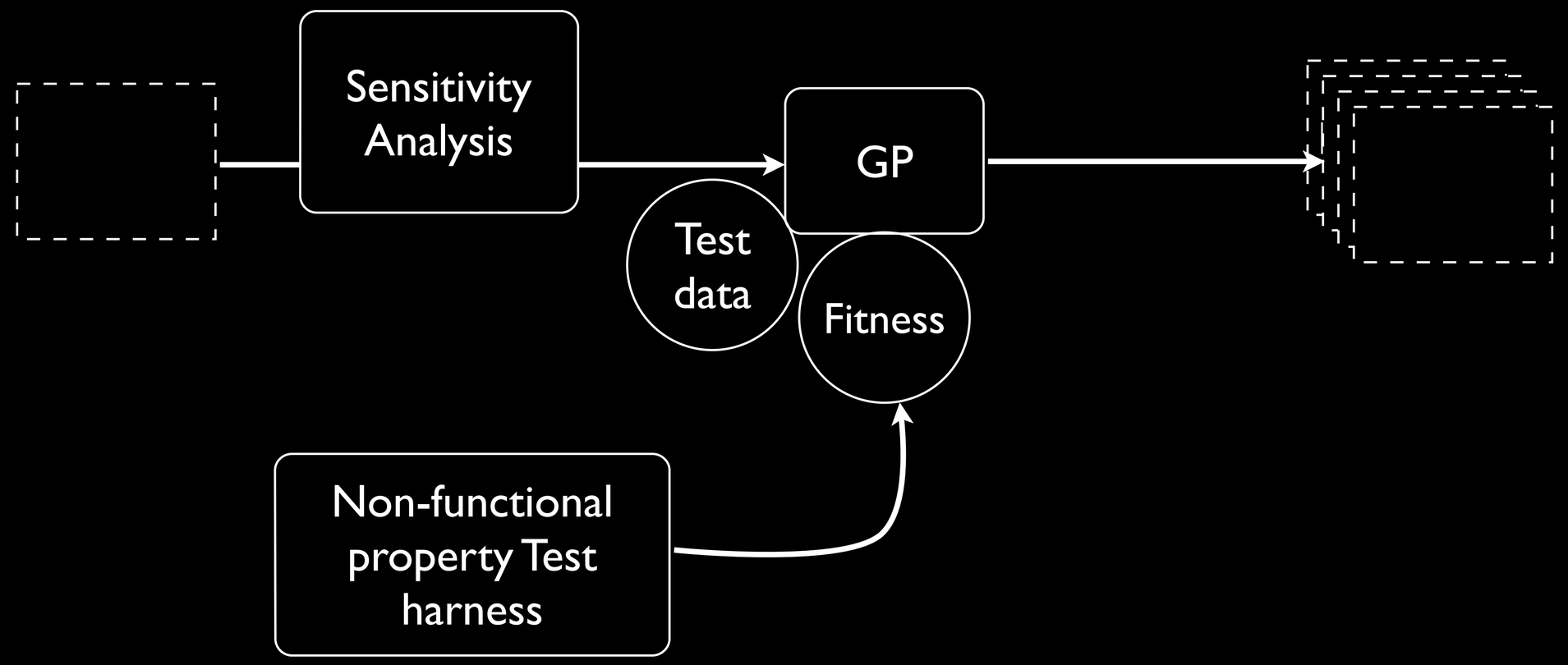


Multi-doner transplant
Specialized for CIT
17% faster

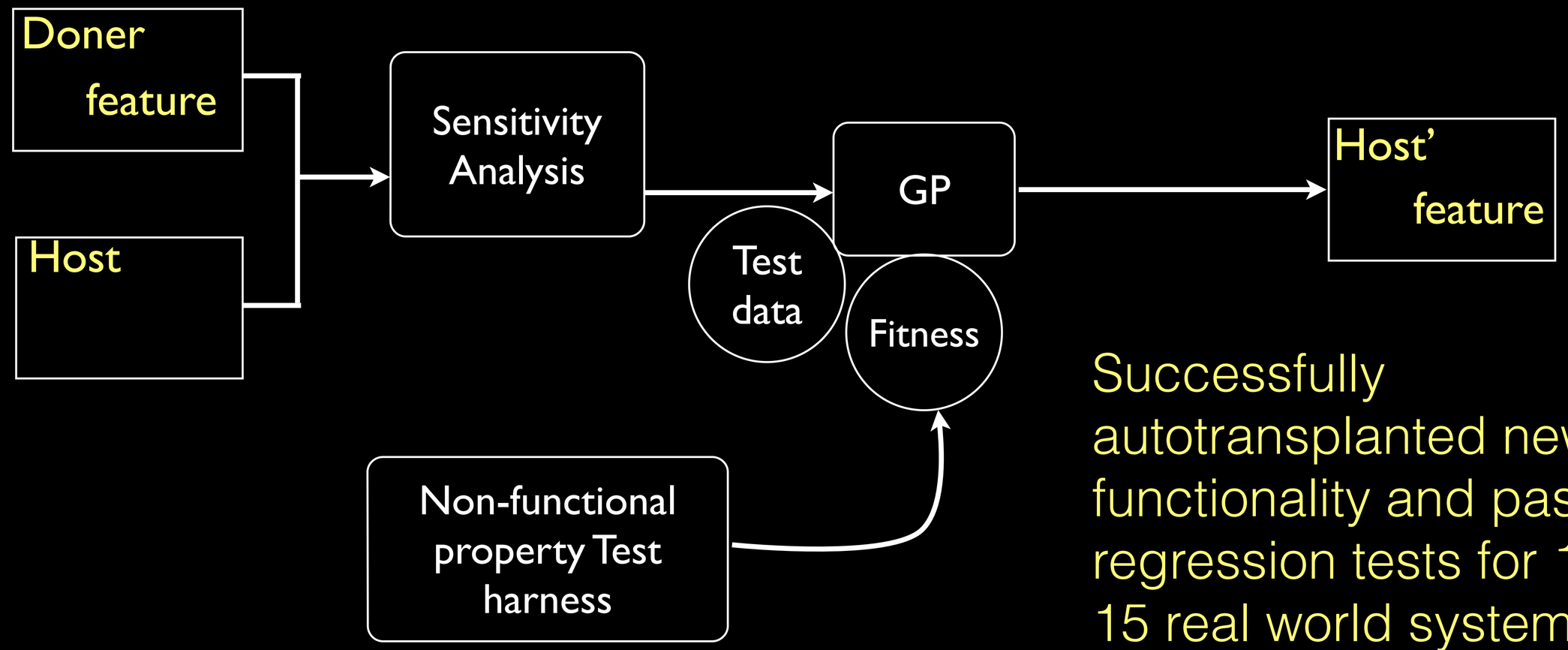
Justyna Petke, Mark Harman, William B. Langdon and Westley Weimer
Using Genetic Improvement & Code Transplants to Specialise a C++ program to a Problem Class (EuroGP'14)

GECCO Humie silver medal

Real world cross system transplantation



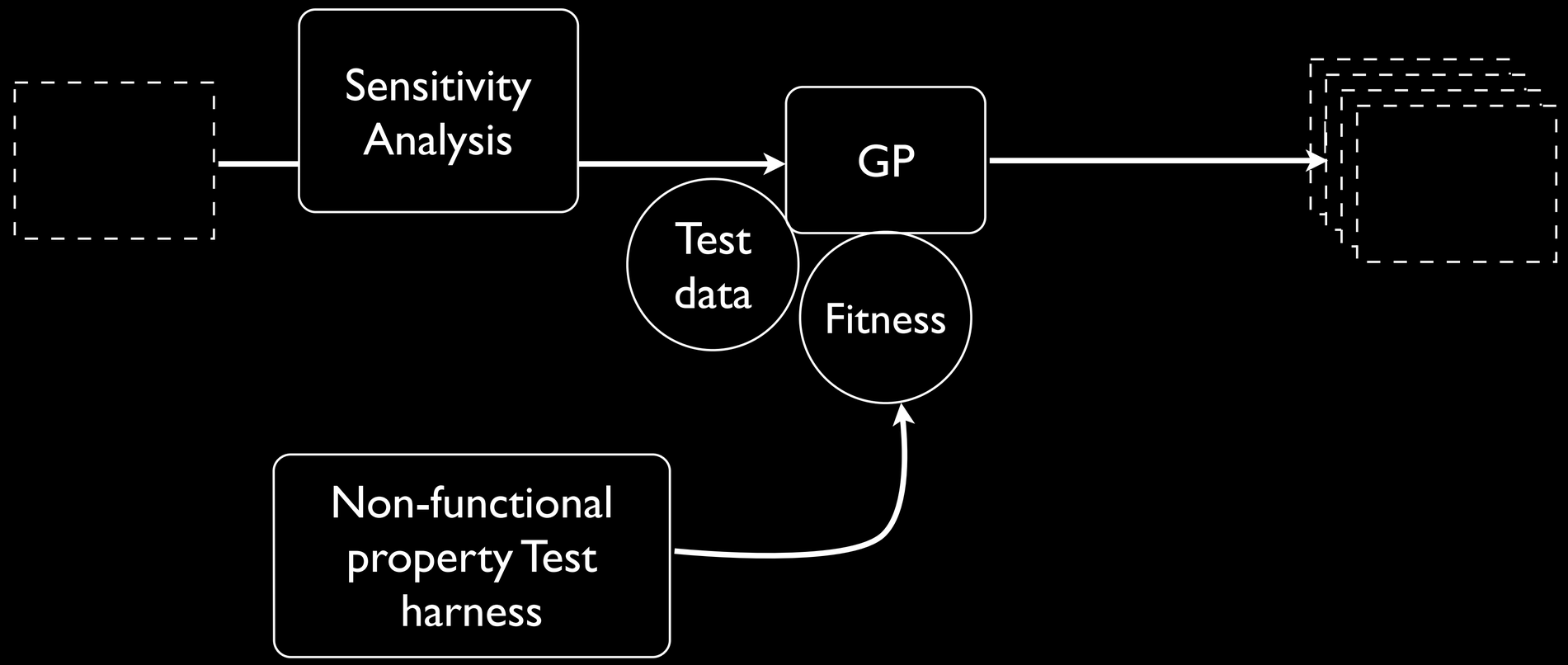
Real world cross system transplantation



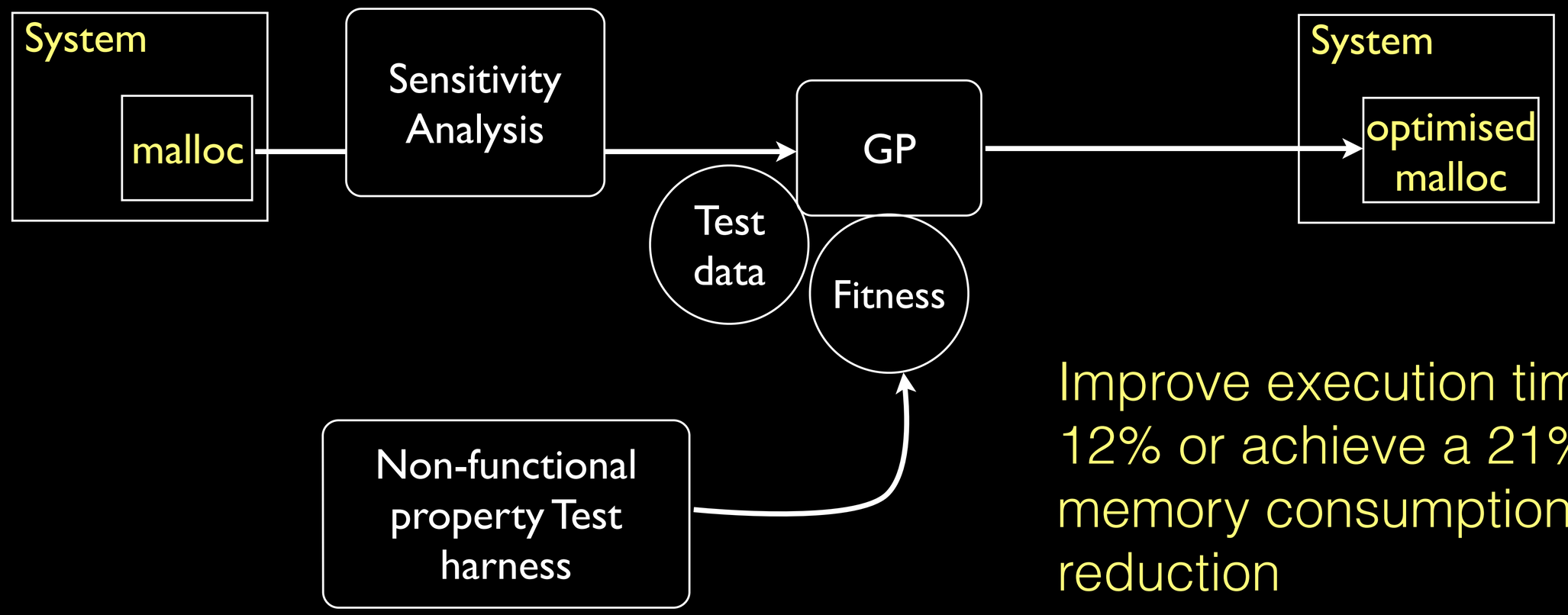
Successfully autotransplanted new functionality and passed all regression tests for 12 out of 15 real world systems

Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke
Automated Software Transplantation (Tech Report). To appear.

Memory speed trade offs



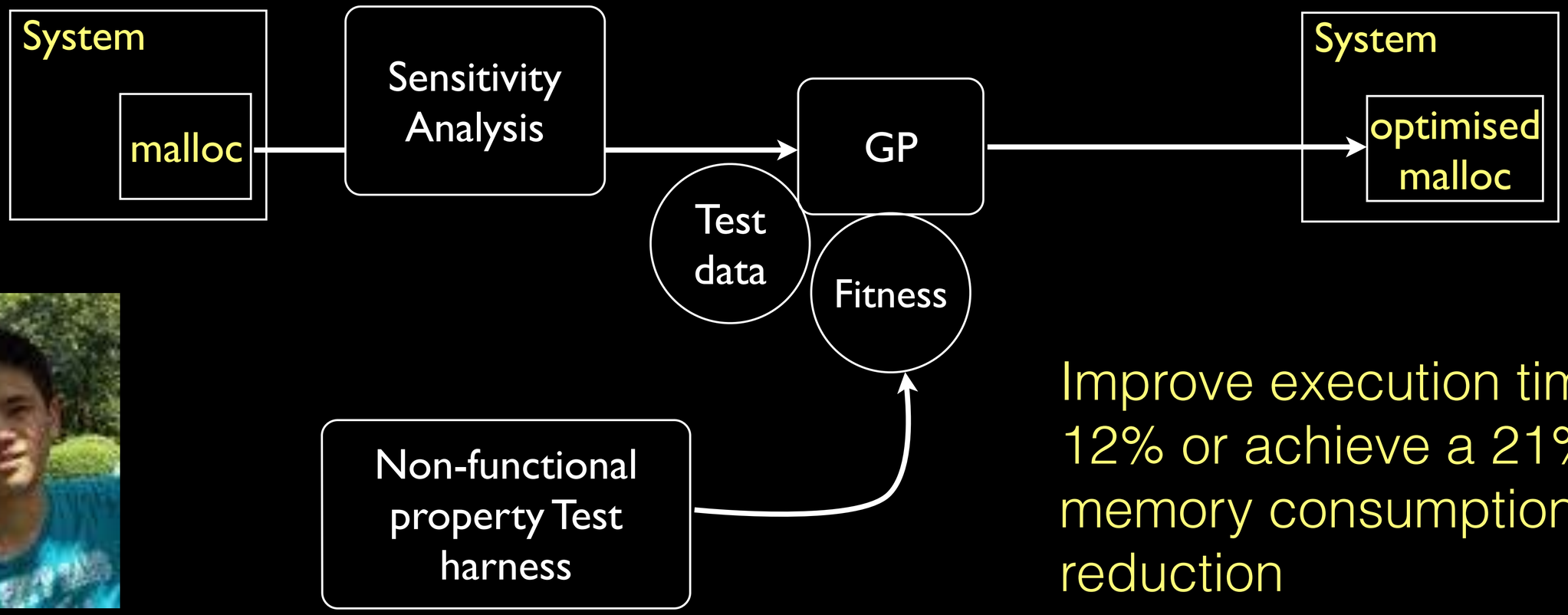
Memory speed trade offs



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), To appear

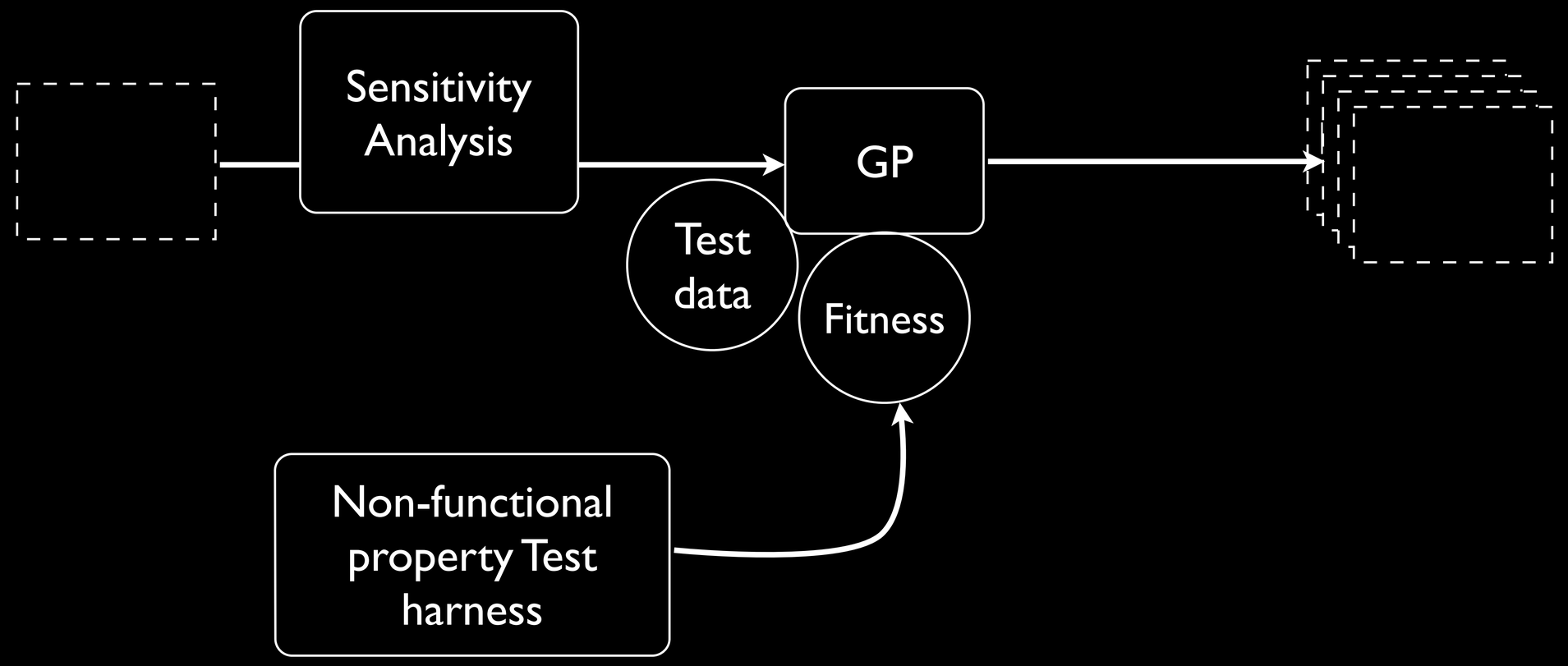
Memory speed trade offs



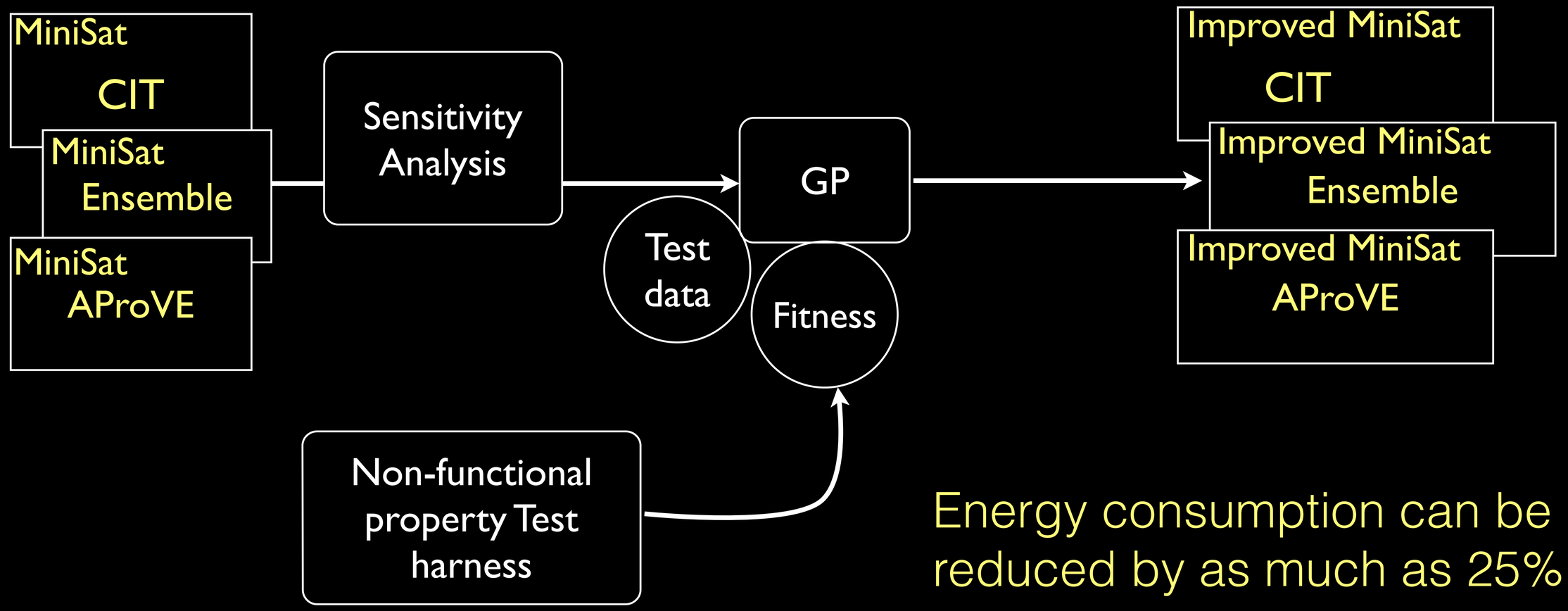
Improve execution time by 12% or achieve a 21% memory consumption reduction

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

Reducing energy consumption



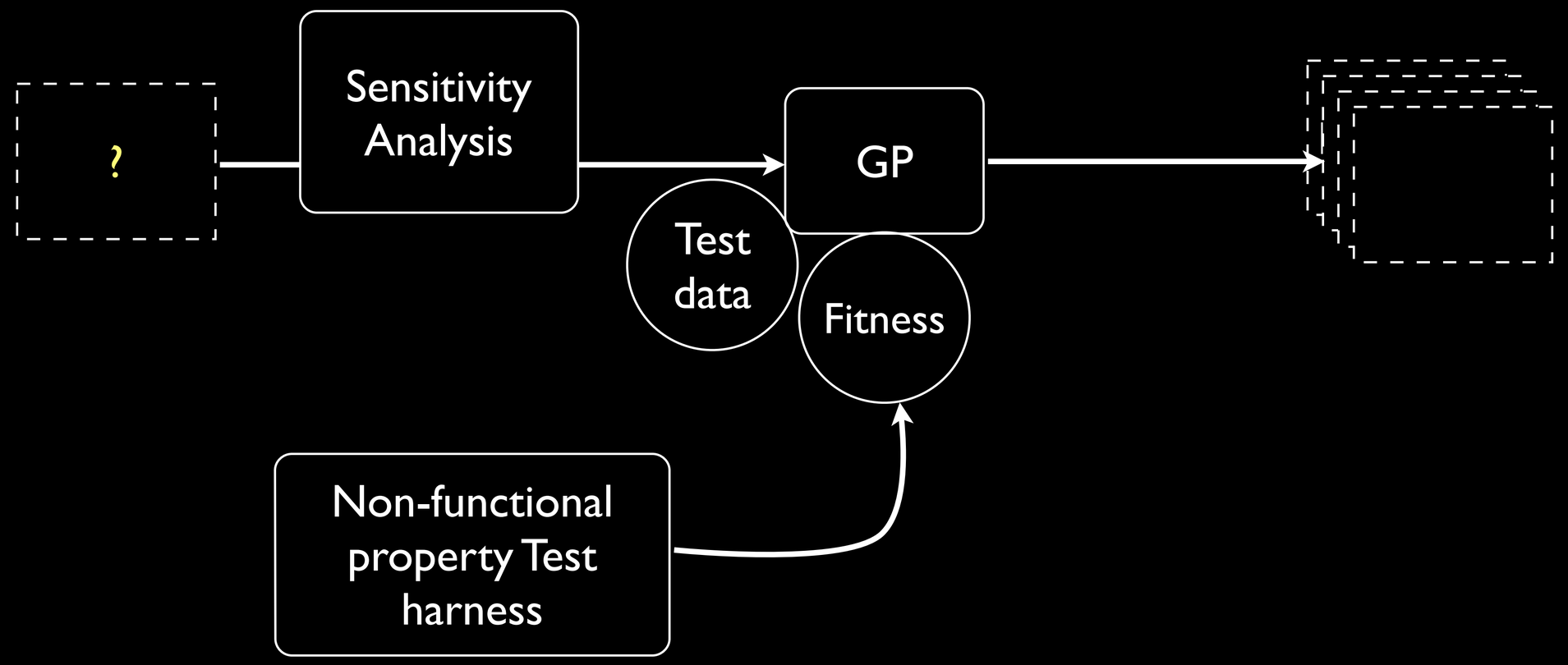
Reducing energy consumption



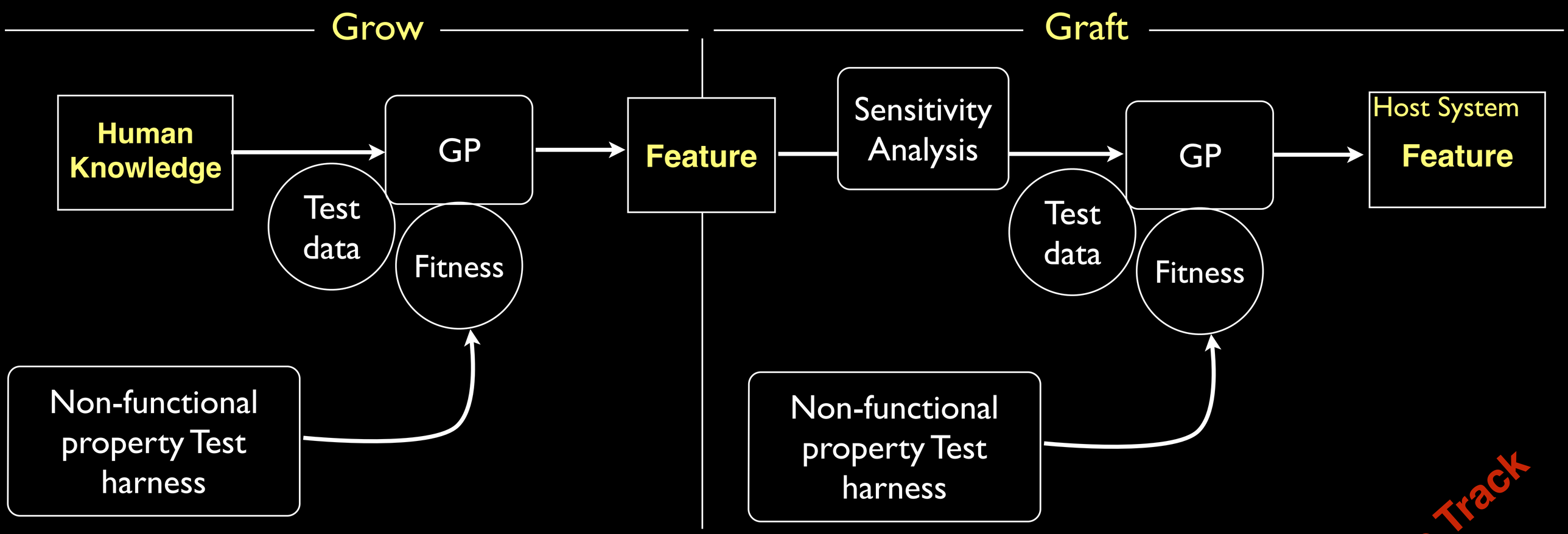
Energy consumption can be reduced by as much as 25%

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



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)

Challenge Track Award

Another keynote paper at ASE 2012

The GISMOE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs*

Mark Harman[1], William B. Langdon[1], Yue Jia[1], David R. White[2], Andrea Arcuri[3], John A. Clark[4]
[1]CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
[2]School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
[3]Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
[4]Department of Computer Science, University of York, Deramore Lane, York, YO10 5GH, 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 differ in their non-functional trade offs. 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

Algorithms, Design, Experimentation, Human Factors, Languages, Measurement, Performance, Verification.

*This position paper accompanies the keynote given by Mark Harman's at the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 12) in Essen, Germany. It is joint work with Bill Langdon, Yue Jia, David White, Andrea Arcuri and John Clark, funded by the EPSRC grants SEBASE (EP/D050863, EP/D050618 and EP/D052785), GISMO (EP/I033688) and DAASE (EP/J017515/) and by EU project FITTEST (257574).

ASE'12, September 3–7, 2012, Essen, Germany.

Keywords

SBSE, Search Based Optimization, 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 themselves making arbitrary choices when such objective 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 in which the system may be deployed.

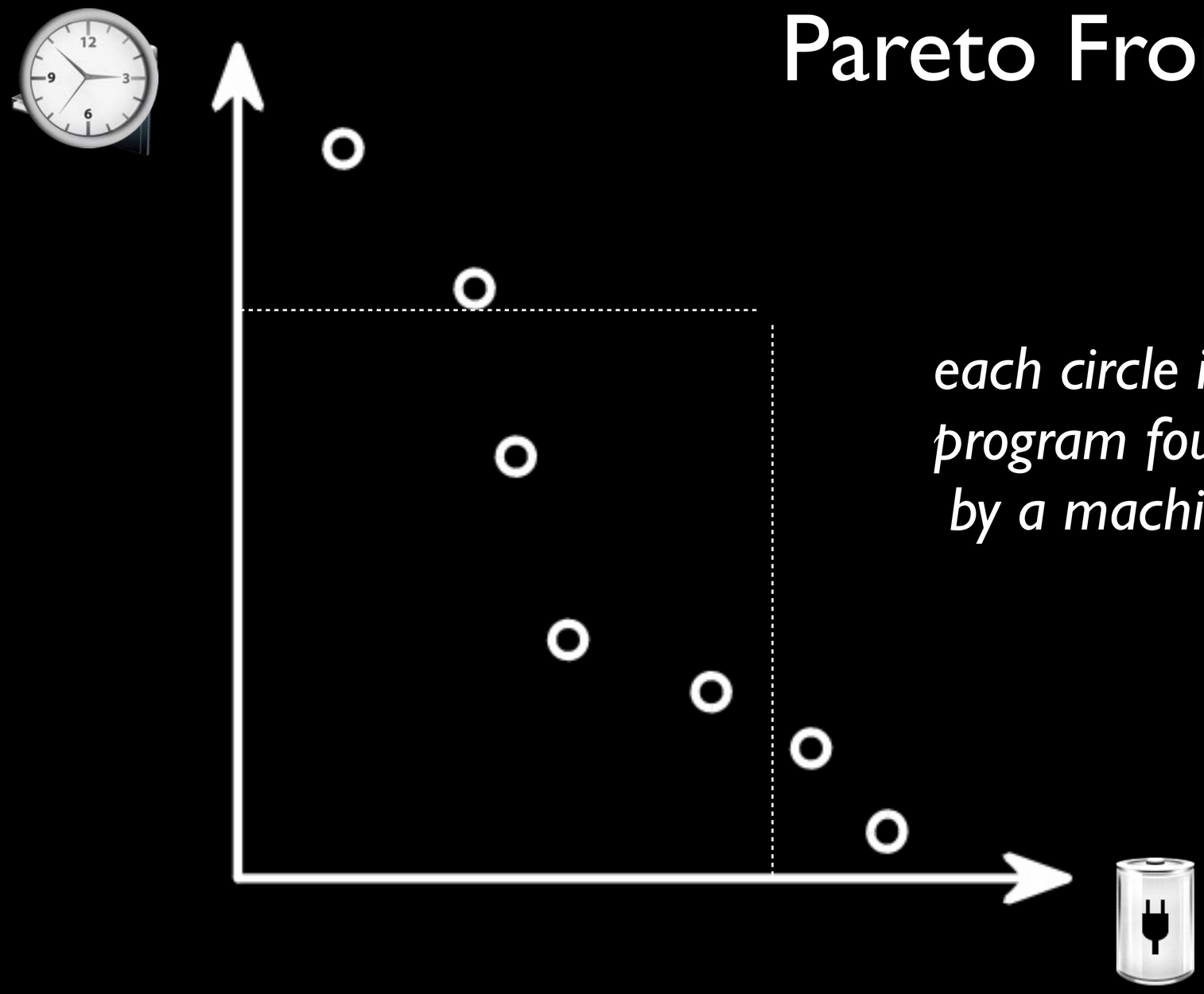
Emergent computing application paradigms require systems that are not only reliable, compact and fast, but which also optimise many different competing and conflicting objectives such as response time, throughput and consumption of resources (such as power, bandwidth and memory). As a result, operational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

Human software developers cannot be expected to optimally balance these multiple competing constraints and may miss 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 conundrum 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.

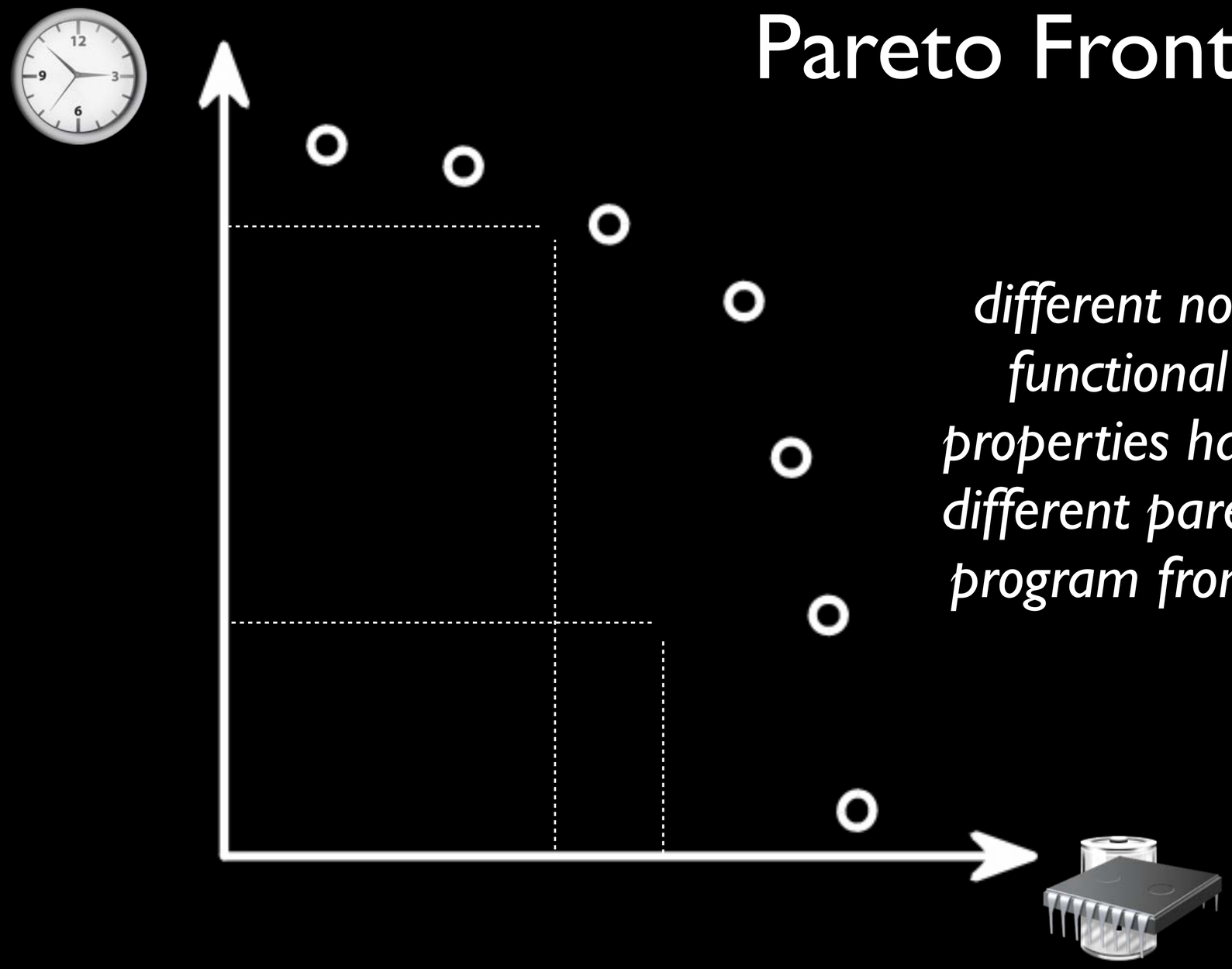
Pareto Front

Pareto Front



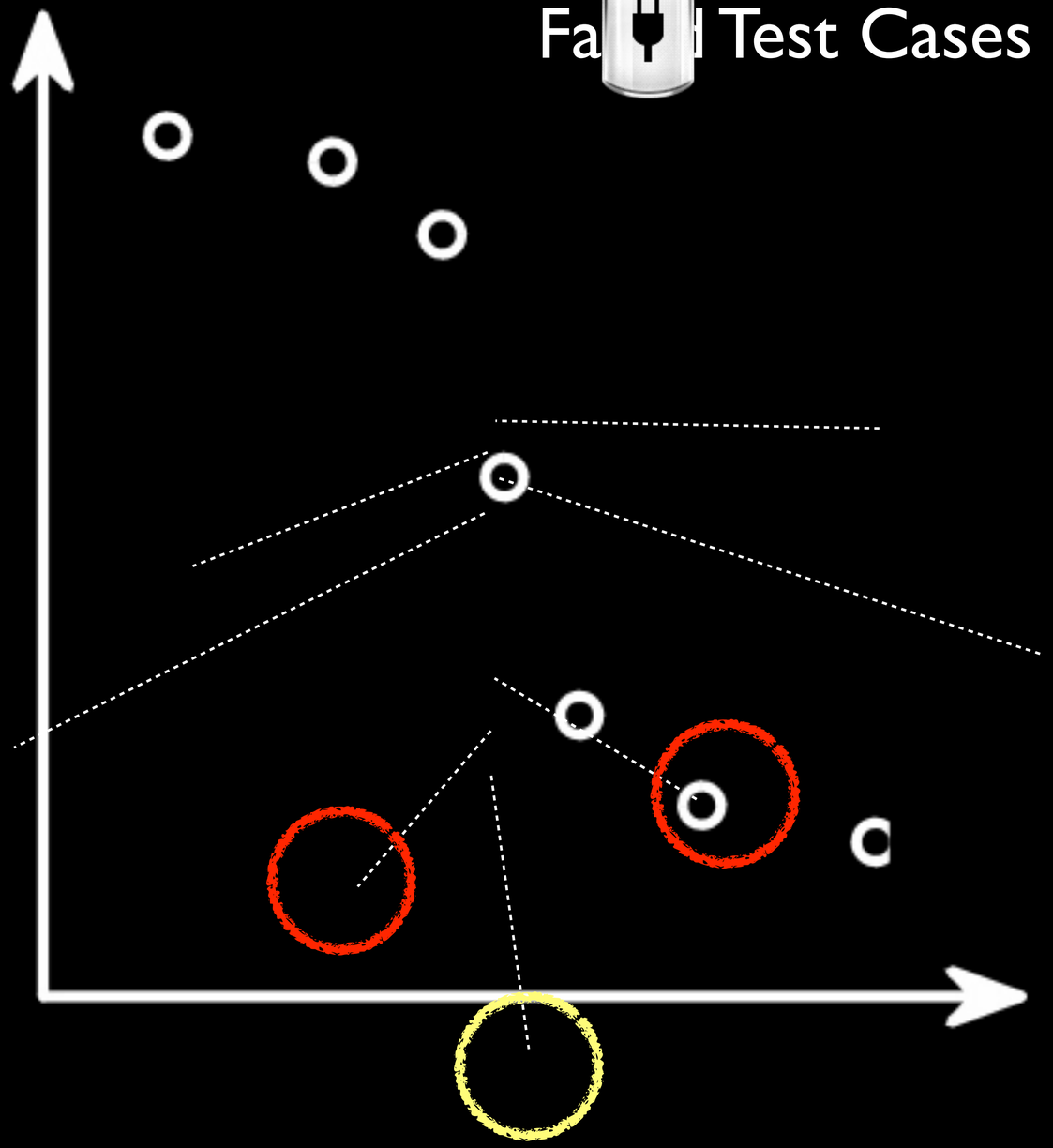
each circle is a program found by a machine

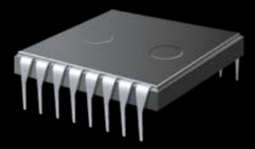
Pareto Front



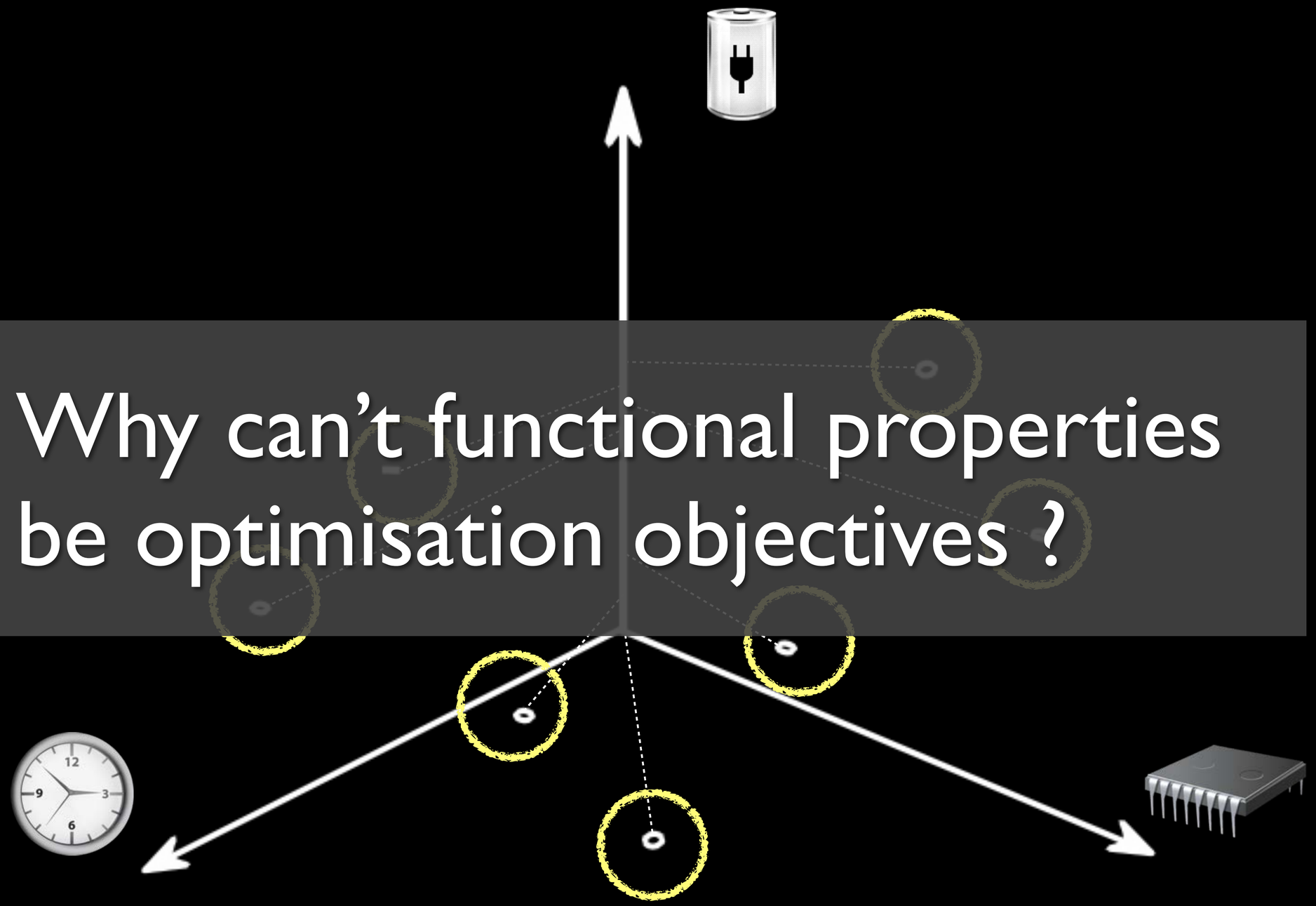
different non functional properties have different pareto program fronts

Fa l Test Cases



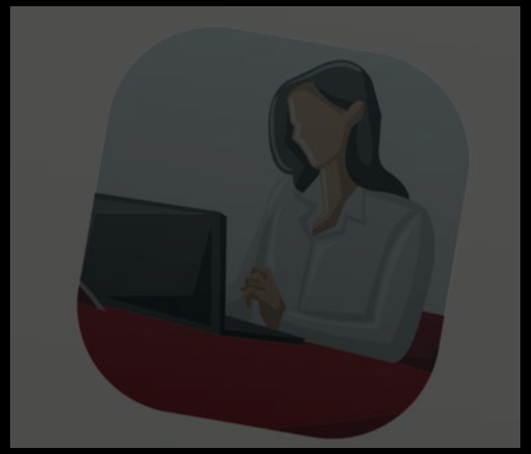


Why can’t functional properties be optimisation objectives ?

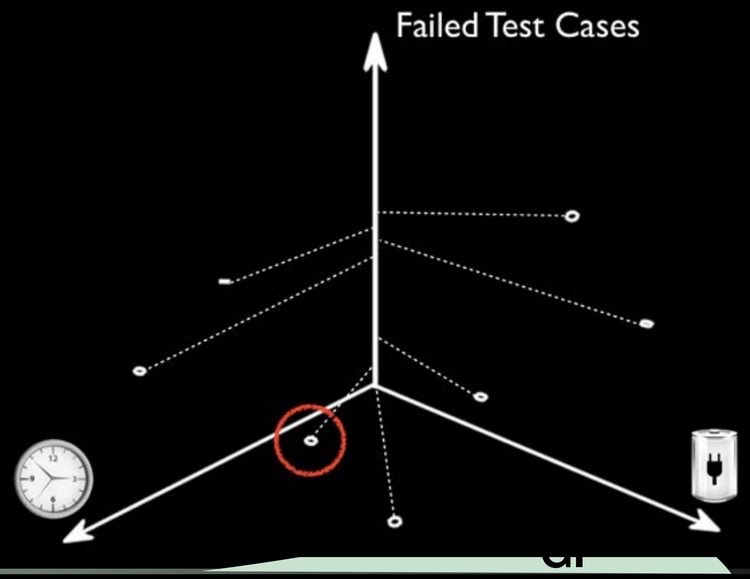


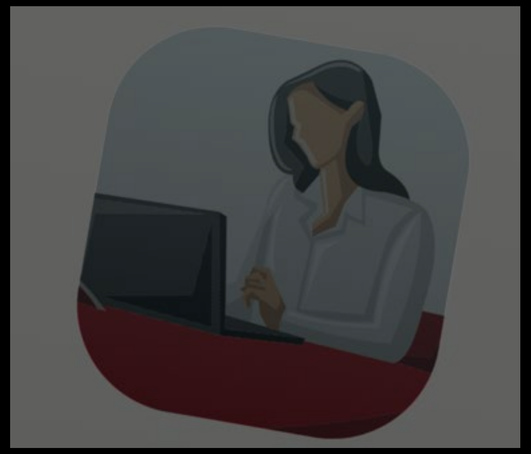
Optimisation

Optimisation

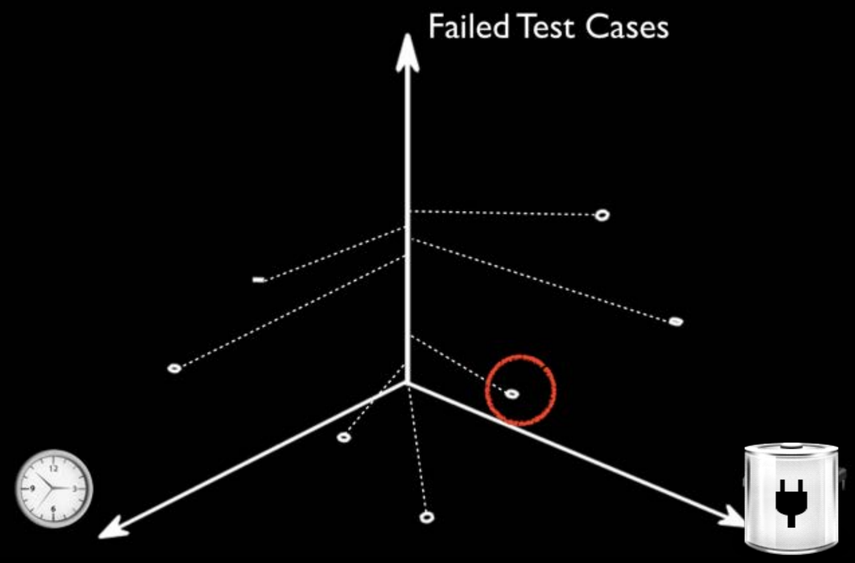


2.5 times faster but
failed 1 test case?

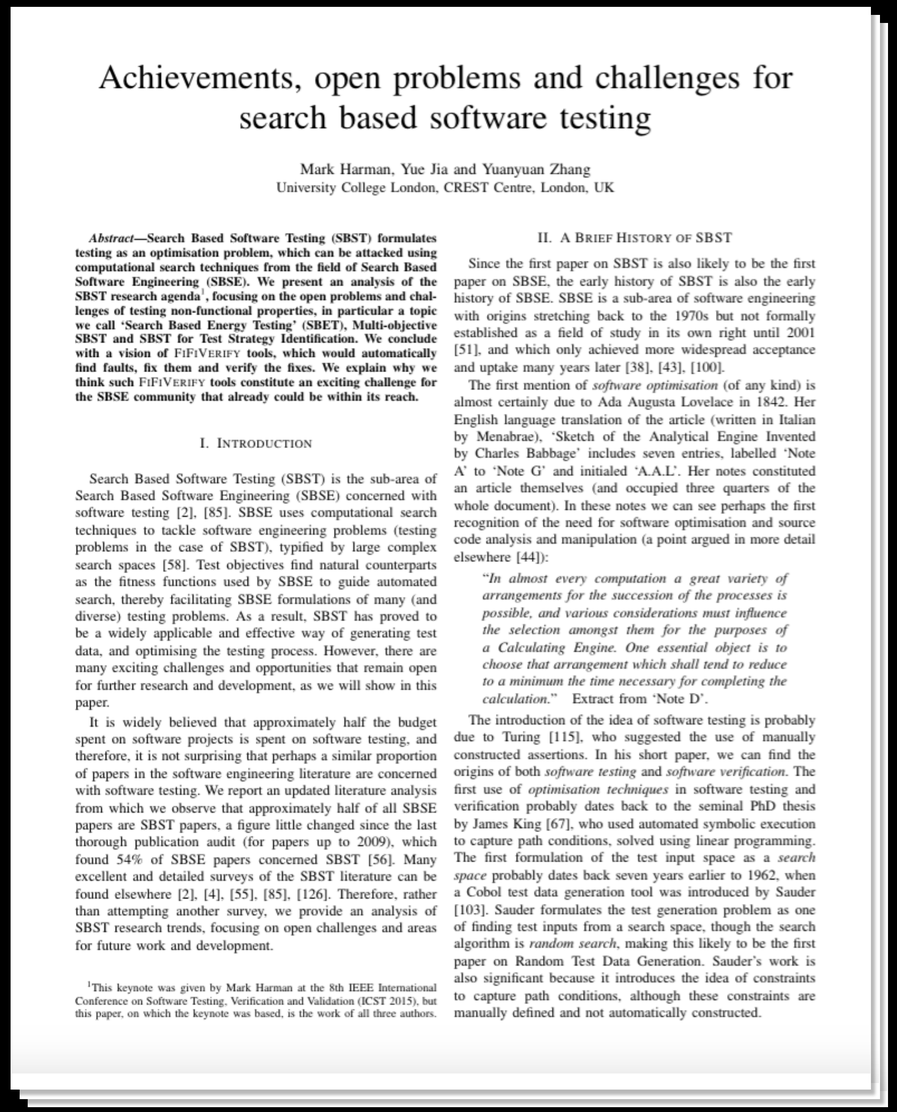




Optimisation



double the battery life
but failed 2 test cases?



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

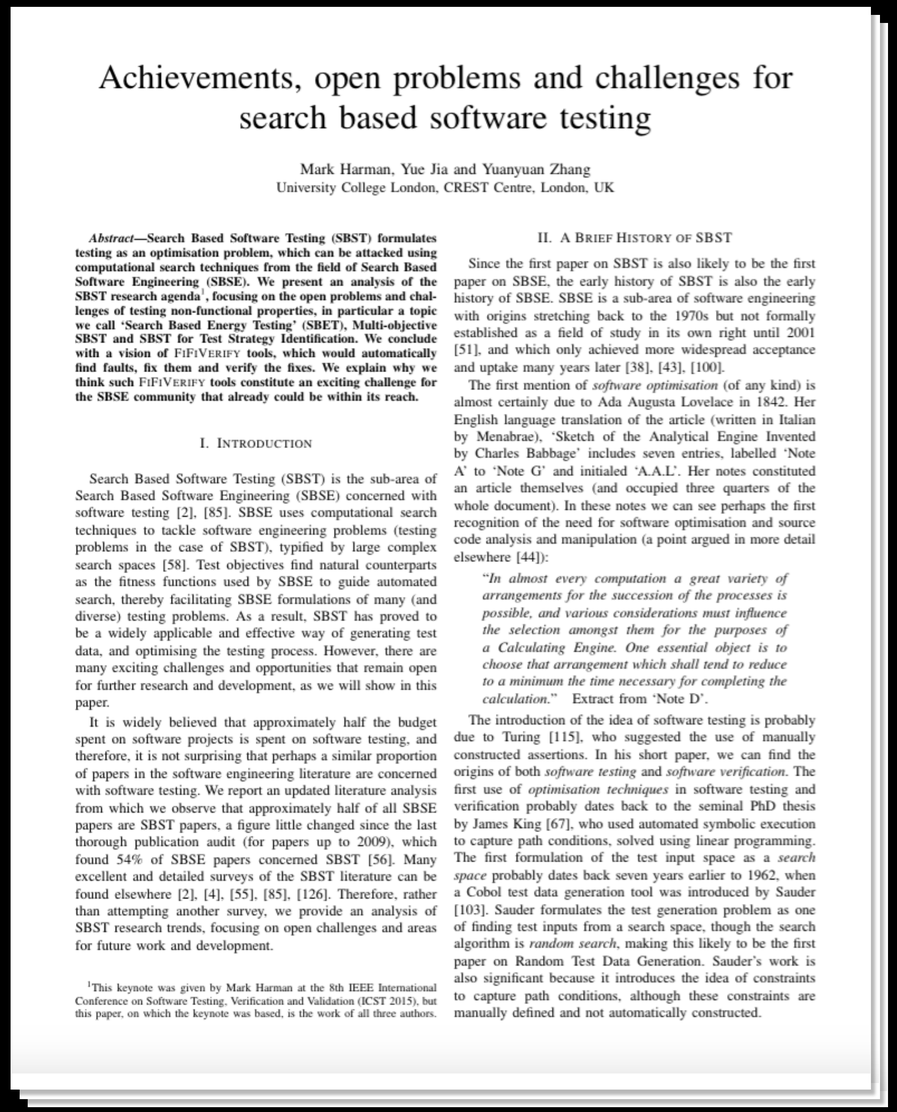
The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both *software testing* and *software verification*. The first use of *optimisation techniques* in software testing and verification probably dates back to the seminal PhD thesis by James King [67], who used automated 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.

Summary

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



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[1], 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 FiFiVerify tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FiFiVerify 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 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 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 [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. 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.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

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 Menabrae), '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]):

> "*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'.

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 automated 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.

Summary

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

Picture Copyrights

http://en.wikipedia.org/wiki/Thomas_Edison#/media/File:Edison_bulb.jpg

http://en.wikipedia.org/wiki/Colossus_computer#/media/File:Colossus.jpg

http://en.wikipedia.org/wiki/Hippie#/media/File:Woodstock-kids.jpg

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