Genetic Improvement for Adaptive Software Engineering

Mark Harman

joint work with Yue Jia, Bill Langdon, Iman Moghadam, Justyna Petke, Shin Yoo & Fan Wu

University College London
COWs

CREST Open Workshop
Roughly one per month

Discussion based
Recorded and archived
COWs

CREST Open Workshop
Roughly one per month

Discussion based
Recorded and archived
COWs

CREST Open Workshop
Roughly one per month
Discussion based
Recorded and archived
COWs
What is SBSE
What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

- potentially exhaustive
- pick one at random

sweet spot
- like google search?
- like code search?
- like breadth first search?
What is SBSE

Search Based Optimization

Software Engineering
What is SBSE

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

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

In SBSE we apply search techniques to search large search spaces, guided by a fitness function that captures properties of the acceptable software artefacts we seek.

- Genetic Algorithms
- Hill Climbing
- Ant Colonies
- Particle Swarm Optimization
- Simulated Annealing
- Genetic Programming
- Greedy
- Estimation of Distribution Algorithms
- Random
- Genetically Informed Search
What is SBSE

let’s listen to software engineers ...

... what sort of things do they say?
Software Engineers Say

We need to satisfy business and technical concerns
We need to reduce risk while maintaining completion time
We need increased cohesion and decreased coupling
We need fewer tests that find more nasty bugs
We need to optimise for all metrics $M_1, ..., M_n$
Software Engineers Say

We need to satisfy business and technical concerns
We need to reduce risk while maintaining completion time
We need increased cohesion and decreased coupling
We need fewer tests that find more nasty bugs
We need to optimise for all metrics $M_1, \ldots, M_n$
Software Engineers Say

- We need to satisfy business and technical concerns
- We need to reduce risk while maintaining completion time
- We need increased cohesion and decreased coupling
- We need fewer tests that find more nasty bugs
- We need to optimise for all metrics $M_1, \ldots, M_n$
Software Engineers Say

We need to satisfy business and technical concerns
We need to reduce risk while maintaining completion time
We need increased cohesion and decreased coupling
We need fewer tests that find more nasty bugs
We need to optimise for all metrics M1,..., Mn
Software Engineers Say

We need to satisfy business and technical concerns

We need to reduce risk while maintaining completion time

We need increased cohesion and decreased coupling

We need fewer tests that find more nasty bugs

We need to optimise for all metrics $M_1, ..., M_n$
Software Engineers Say

- We need to satisfy business and technical concerns
- We need to reduce risk while maintaining completion time
- We need increased cohesion and decreased coupling
- We need fewer tests that find more nasty bugs
- We need to optimise for all metrics M₁, ..., Mₙ
Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1,...,Mn
Software Engineers Say

Requirements: We need to satisfy business and technical concerns

Management: We need to reduce risk while maintaining completion time

Design: We need increased cohesion and decreased coupling

Testing: We need fewer tests that find more nasty bugs

Refactoring: We need to optimise for all metrics M1, ..., Mn

All have been addressed in the SBSE literature
Growth Trends
Polynomial rise in publications
Author statistics

more than 1250 authors
more than 1150 papers
more than 390 institutions
more than 50 countries

source: SBSE repository, July 2013.
Software Engineering topics attacked
Software Engineering topics attacked
Just some of the many SBSE applications

Agent Oriented
Aspect Oriented
Assertion Generation
Bug Fixing
Component Oriented
Design
Effort Estimation
Heap Optimisation
Model Checking
Predictive Modelling
Probe distribution
Program Analysis
Program Comprehension
Program Transformation
Project Management
Protocol Optimisation
QoS
Refactoring
Regression Testing
Requirements
Reverse Engineering
SOA
Software Maintenance and Evolution
Test Generation
UIO generation
EPSRC
Grant
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014

Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

EPSRC Grant 2012-2018 PhD and post docs offered

Stirling

York

Birmingham

UCL

Mark.Harman@ucl.ac.uk
SBSE Tutorial and Survey
SBSE Tutorial and Survey

SBSE Tutorial and Survey


SBSE Tutorial and Survey


google: SBSE tutorial

SBSE Tutorial and Survey


google: SBSE tutorial


google: SBSE survey
SBSE Tutorial and Survey

Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza and Shin Yoo. 
Search Based Software Engineering: Techniques, Taxonomy, Tutorial. 

google: SBSE tutorial

Mark Harman, Afshin Mansouri and Yuanyuan Zhang. 
Search Based Software Engineering: Trends, Techniques and Applications 
ACM Computing Surveys. 

google: SBSE survey
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014

Modularisation
Search based modularisation

Mancoridis and Mitchell: IWPC 1998 and TSE 2006
Praditwong, Harman and Yao: TSE 2011
Barros: GECCO 2012
Search based modularisation

Mancoridis and Mitchell: IWPC 1998 and TSE 2006
Praditwong, Harman and Yao: TSE 2011
Barros: GECCO 2012
Search based modularisation

Mancoridis and Mitchell: IWPC 1998 and TSE 2006
Praditwong, Harman and Yao: TSE 2011
Barros: GECCO 2012

Take home: multi objective search helps even single objective problems
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014

Next Release Problem
Search based requirements

Bagnall, Rayward-Smith and Whittley: IST 2001
Zhang, Harman and Mansouri: GECCO 2007
Saliu and Ruhe: FSE 2007
Search based requirements

Bagnall, Rayward-Smith and Whittley: IST 2001
Zhang, Harman and Mansouri: GECCO 2007
Saliu and Ruhe: FSE 2007

Take home: SBSE is well suited to cost value trade offs
Search Based Management
Search based management

Dolado: IST 2001
Chicano and Alba: MIC 2005
Ferrucci, Harman, Ren and Sarro: ICSE 2013
Search based management

Dolado: IST 2001
Chicano and Alba: MIC 2005
Ferrucci, Harman, Ren and Sarro: ICSE 2013

Take home: SBSE can help analyse risk - reward
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014

- Design
- Requirements
- Management
- Debugging
- Testing
- Verification
- Maintenance

...
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014
Search based repair

Arcuri and Yao: CEC 2008
Weimer, Nguyen, Le Goues and Forrest: ICSE 2009
Kim, Nam, Song and Kim: ICSE 2013
Search based repair

Arcuri and Yao: CEC 2008
Weimer, Nguyen, Le Goues and Forrest: ICSE 2009
Kim, Nam, Song and Kim: ICSE 2013

Take home: some (~50%) real bugs are easy to fix
Take home: SBSE loves testing problem: adequacy = fitness
<table>
<thead>
<tr>
<th>SB</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GA</td>
</tr>
<tr>
<td></td>
<td>GP</td>
</tr>
<tr>
<td></td>
<td>ES</td>
</tr>
<tr>
<td></td>
<td>SA</td>
</tr>
<tr>
<td></td>
<td>HC</td>
</tr>
<tr>
<td></td>
<td>ACO</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

- Design
- Requirements
- Management
- Debugging
- Testing
- Verification
- Maintenance
- ...
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014

- Design
- Requirements
- Management
- Debugging
- Testing
- Verification
- Maintenance
- ...

SB

SE

GA
GP
ES
SA
HC
ACO
...

Test suite optimisation
Test suite optimisation

Yoo and Harman: ISSTA 2007
Li, Harman and Hierons: TSE 2007
Mirarab, Akhlaghi and Tahvildari: TSE 2012
Test suite selection

Yoo and Harman: ISSTA 2007
Li, Harman and Hierons: TSE 2007
Mirarab, Akhlaghi and Tahvildari: TSE 2012
Test suite selection

Find faults faster

Optimise faster using GPGPU

Take home: regression testing is all about optimisation
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST
Design
Requirements
Management
Debugging
Testing
Verification
Maintenance
...

SB

SE

GA  GP  ES  SA  HC  ACO  ...

optimising model checking

Take home: Model checkers search large spaces
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SB

SE

Design
Requirements
Management
Debugging
Testing
Verification
Maintenance
...

Search based Refactoring

GA  GP  ES  SA  HC  ACO  ...

Search based Refactoring
Search based refactoring

O’Keeffe and Ó Cinnéide: CSMR 2003
Harman and Tratt: GECCO 2007
Jensen and Cheng: GECCO 2010
Search based refactoring

O’Keeffe and Ó Cinnéide: CSMR 2003
Harman and Tratt: GECCO 2007
Jensen and Cheng: GECCO 2010

Take home: refactoring is a multi objective optimisation problem
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014

SE

SB

- Design
- Requirements
- Management
- Debugging
- Testing
- Verification
- Maintenance
- ...

GA  GP  ES  SA  HC  ACO  ...

Program Repair
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014
Gismoe

Pareto program surface is automatically constructed to support dialog with the software designer concerning trade offs in the solution space of programs

What is a pareto program surface?
The GismoE challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs

Mark Harman¹, William B. Langdon², Yue Jia³, David R. White³, Andrea Arcuri⁴, John A. Clark⁴
¹CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
²School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
³Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
⁴Department 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

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.
The GISMEO challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs.

Mark Harman¹, William B. Langdon¹, Yue Jia¹, David R. White², Andrea Arcuri³, John A. Clark⁴
¹CREST Centre, University College London, Gower Street, London, WC1E 6BT, UK.
²School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland, UK.
³Simula Research Laboratory, P. O. Box 134, 1325 Lysaker, Norway.
⁴Department 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 compilers and tools. Rather than focusing on the compilation step, the designer focuses on the exploration of the program space itself.

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 cater to, with challenges such as bandwidth, energy efficiency, and user experience. A single compilation step will not be enough; a programming environment that allows the designer to explore multiple solutions will be required.

Therefore, we propose a new kind of software development environment that automatically generates a set of candidate programs, allowing the software designer to navigate the Pareto surface of non-functional trade-offs and select the best solution for their particular requirements.
programming is changing
Functional Requirements

Non-Functional Requirements

Requirements
Functional Requirements

- functionality of the Program

Non-Functional Requirements

- Execution Time
- Memory
- Bandwidth
- Battery
- Size
Software Design Process
Software Design Process
Software Design Process
Software Design Process
Software Design Process
Multiplicity

- Multiple Platforms
- Conflicting Objectives
- Multiple Devices

Genetic Improvement for Adaptive Software Engineering
Mark Harman, UCL CREST
Why is the programmer human?
Which requirements must be human coded?

Functional Requirements:
- humans have to define these

Non-Functional Requirements:
- a machine can optimise these
Which requirements are essential to human?

Functional Requirements
humans have to define these

Non-Functional Requirements
a machine can optimise these
Pareto Front
Pareto Front

each circle is a program found by a machine
different non-functional properties have different pareto program fronts
Why can’t functional properties be optimisation objectives?
Optimisation
Optimisation

2.5 times faster but failed 1 test case?
Optimisation

do double the battery life but failed 2 test cases?
Genetic Programming for Software Transplantation
Genetic Programming for Software Transplantation
covered in more detail in the WCRE 2013 keynote paper
GP for Transplants
GP for Transplants
GP for Transplants
GP for Transplants

Localise

\textbf{A}bstract

\textbf{T}arget

\textbf{I}nterface

\textbf{I}nsert

\textbf{V}alidate

\textbf{R}epeat

\textbf{-------------
-------------
-------------
-------------
-------------

\textbf{Doner}

\textbf{if (ptr! = NULL)}
\textbf{foo (*ptr);}

\textbf{-------------
-------------
-------------
-------------
-------------

\textbf{Host}
GP for Transplants

Localise

Abstract

if (ptr != NULL)
    foo(*ptr);

if (   != NULL)
    (    );
GP for Transplants

Localise

Abstract

Target

I

II

III

IV

R

Doner

SBSE

Host
GP for Transplants

Localise

Abstract

Target

Interface

Insert

Validate

Repeat

Doner

Host

Interface

Target
GP for Transplants

Localise

Abstract

Target

Interface

Insert

Validate

Repeat

if ( □_1 != NULL)
□_2 ( □_1 );

Doner

Host

Interface

Target
GP for Transplants

Localise

Abstract

Target

Interface

Insert

\[
\text{if}\left(\text{host}\_\text{ptr}!=\text{NULL}\right)
\]

\[
\text{host}\_\text{action}(\text{*host}\_\text{ptr});
\]
GP for Transplants

**Localise**

**Abstract**

**Target**

**Interface**

**Insert**

1 new feature tests

**Validate**

2 regression tests

**R**

3 quality tests
GP for Transplants

Localise

Abstract

Target

Interface

Insert

Validate

Repeat

Doner

Host
Offline Genetic Improvement
Offline Genetic Improvement

... what have we managed to achieve so far ...
Genetic Improvement of Programs

- Bowtie2
- Sensitivity Analysis
- GP
- Test data
- Non-functional property Test harness
- Fitness
- Improved Bowtie2

- 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 2014 (TR available)
Genetic Improvement of Programs

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)
Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

Program Improvement

- Design
- Management
- Debugging
- Testing
- Verification
- Maintenance
- Requirements
- Maintenance
- ...
Genetic Improvement
Babel Pidgin: Grow new functionality

Growing

GP

Test data

Fitness

Bi-language translation

Grafting

...

안녕하세요
Olá

hello

Mark Harman, Yue Jia and William B Langdon
Babel Pidgin: SBSE can grow and graft entirely new functionality into a real world system (SSBSE’14 Challenge)
Online Genetic Improvement
Online Genetic Improvement

This is described in the SEAMS 2014 keynote paper
Online phase

Offline phase

User specify operation characteristics
Online phase

User specify operation characteristics

Environmental and usage profile Learning

Offline phase

Data Collection

Phase 1

User specify operation characteristics
Online phase

User specify operation characteristics

Offline phase

Environmental and usage profile Learning

Tuneable Parameters

Phase 1

Data Collection

Program
Online phase

User specify operation characteristics

Offline phase

Environmental and usage profile Learning

Tuneable Parameters

Program

Tuneable Implicit Parameters

Exposing Implicit Parameters

Phase 1

Data Collection
Online phase

User specify operation characteristics

Phase 1
Data Collection

Offline phase

Environmental and usage profile Learning

Program

SBSE

Optimisation Tuning
Online phase

User specify operation characteristics

Phase 1

Data Collection

Environmental and usage profile Learning

Program

Offline phase

Patch generation

Optimisation Tuning

Genetic Improvement for Adaptive Software Engineering

Mark Harman, UCL CREST

SEAMS2014
Online phase

- User specifies operation characteristics

Offline phase

- Environmental and usage profile learning
- Patch generation
- Optimisation and tuning

Phase 1
- Data Collection

Phase 1-n
- Deploy patches
- Data Collection

Program

SBSE
Online phase

User specify operation characteristics

Phase 1
Data Collection

Phase 1- n

Deploy patches
Data Collection

Offline phase

Environmental and usage profile Learning

Program

SBSE

Patch generation

Optimisation Tuning

Genetic Improvement for Adaptive Software Engineering
Mark Harman, UCL CREST
Example: Wikipedia Mobile
Example: Wikipedia Mobile
Example: Wikipedia Mobile

Daily Operating Mode
Example: Wikipedia Mobile

Daily Operating Mode
Example: Wikipedia Mobile

Daily Operating Mode
Example: Wikipedia Mobile
Example: Wikipedia Mobile

Overnight Charging Mode

GP to search for caching strategies
Online Genetic Improvement

... could provide an SBSE for Self Adaptive Systems

Genetic Improvement
  +
Grow and Graft Patches
  +
Expose parameters and Autotune
  +
Learning and deployment: Catch, Dream, Optimise

More details in keynote papers: ASE12, WCRE13 & SEAMS14