# ERC Advanced Grant 2016
# (public version of funded proposal)
# Evolving Program Improvement Collaborators

# EPIC

**Mark Harman**
**Principal Investigator**
University College London
Project duration: 60 months

## Summary

EPIC will automatically construct Evolutionary Program Improvement Collaborators (called Epi-Collaborators) that suggest code changes that improve software according to multiple functional and non-functional objectives. The Epi-Collaborator suggestions will include transplantation of code from a donor system to a host, grafting of entirely new features 'grown' (evolved) by the Epi-Collaborator, and identification and optimisation of tuneable 'deep' parameters (that were previously unexposed and therefore unexploited).

A key feature of the EPIC approach is that all of these suggestions will be underpinned by automatically-constructed quantitative evidence that justifies, explains and documents improvements. EPIC aims to introduce a new way of developing software, as a collaboration between human and machine, exploiting the complementary strengths of each; the human has domain and contextual insights, while the machine has the ability to intelligently search large search spaces. The EPIC approach directly tackles the emergent challenges of multiplicity: optimising for multiple competing and conflicting objectives and platforms with multiple software versions.

**Keywords and phrases:** Search Based Software Engineering (SBSE);
Evolutionary Computation,
Software Testing;

## Part B1a: Extended Synopsis of the Scientific Proposal

**The Multiplicity Problem**: Software Engineers increasingly find themselves balancing competing and conflicting operational objectives such as speed, size, response time, bandwidth, and energy consumption. It is far from obvious how to balance these objectives, while maintaining functional correctness, especially given the need to support a multiplicity of platforms, environments and requirement sets. This is the 'multiplicity problem' [28]. We cannot tackle multiplicity using current human-intensive software development techniques, which are becoming expensive and are ultimately unscalable; a significant leap in automation is essential to address the multiplicity problem.

**The Proposed Solution**: The key idea is that evolutionary computation can evolve software improvement collaborators; automated tools that offer specifically-evolved, explained and justified advice on improvements that meet multiple non-functional requirements, while maintaining and extending functionality.

We think of them as 'collaborators' in the sense that they collaborate with the human engineer, maximising the value of their complementary skill set. Humans have insights and domain knowledge inaccessible to machines. By contrast, the machine has the ability to intelligently search large spaces, guided by fitness and can provide reliable experimental data to back up its advice at a scale, breadth and precision unmatchable by humans. To exploit these complementary skills, EPIC will develop theory and algorithms that will automatically construct, from an existing system, sets of improvements, proposed to the human, underpinned by quantitative data that justifies and explains each improvement proposed.

**Illustrative Examples of the Epi-Collaborators**: Suppose the software engineer wants to develop a media player for a mobile device, and to minimise energy consumption to extend battery life while doing so. Using EPIC, the engineer consults specifically-evolved Epi-Collaborators that provide a set of recommended software edits, transplants, and prototype features, backed by evidence from automated experimentation, conducted by the Epi-Collaborators themselves.

The parameter-exposing Epi-Collaborator might advise "expose local variable `bar` and supply actual parameter 42 to `bar`, in function call `foo` at line `3425`". The grow-and-graft Epi-Collaborator might offer an entirely new fragment of code that implements a compression routine for the mobile platform, using an existing desktop compression implementation as a test oracle. Finally, the transplantation Epi-Collaborator could help further, by offering a transplantation of a critical coder-decoder (codec) from a separate donor system into the new mobile device version.

**Evidence for Feasibility**: It will undoubtedly be a technical challenge to provide automatically-evolved collaborators that offer such advice. Nevertheless, there is strong evidence that what I propose is feasible: my team and I have recently demonstrated proof-of-concept solutions that have, indeed, exposed and optimised deeply-embedded and previously hidden parameters, and optimised values that satisfy multiple objectives [141], reduced energy consumption (by 25%) [26], grown-and-grafted compression and other routines [67, 83, 84], and transplanted features into real-world media players [**?**].

However, although this initial work has attracted a great deal of attention and won many awards [**?**, 67, 113], it merely scratches the surface of what is ultimately possible. EPIC will transform these existing glimpses of automated system improvement into a theory and practice of Epi-Collaboration.

**The Importance of Epi-Collaborator Evidence, Justification and Explanation**: We need techniques that justify and explain improvements. Instead of simply proposing to the engineer "supply actual parameter 42" (as might be achieved now), a more sophisticated Epi-Collaborator would offer something more in the nature of "When the global configuration parameter `video-mode` is set to `hi-res`, then 42 is the value of the local variable `bar` that yields the observed minima for the equation *energy consumed* = `foo(pixel-res/bar)`. This edit passes all test cases, and reduces energy consumption by 5%-8% (95% confidence interval)". This goes beyond merely "telling the engineer what to do". It explains *why*.

**How EPIC Addresses the Multiplicity Problem**: Coping with multiplicity involves transplanting and adapting existing code for multiple platforms, identifying and optimising aspects of the code for each such platform, and developing new features to exploit the opportunities offered by each platform. Currently, all of these activities are painfully human-intensive. EPIC proposes a quantum leap in automation, relieving the human engineer of much of this work, which is tedious yet requires both intelligence and tenacity. Together, therefore, the human and the Epi-Collaborators will be more efficient and more effective.

## State of the Art

EPIC uses an approach to Software Engineering, known as Search Based Software Engineering (SBSE). SBSE is the name given to a body of scientific work in which search based optimisation is applied to software engineering [58, 59, 69]. The aim of SBSE research is to move software engineering problems from human-based search to machine-based search. The philosophy that underpins all applications of SBSE (and which also underpins the EPIC project) is to exploit human creativity and machines' tenacity and reliability, rather than requiring humans to perform the more tedious, error-prone (and thereby costly) aspects of the engineering process. SBSE can also provide insights and decision support [3, 69].

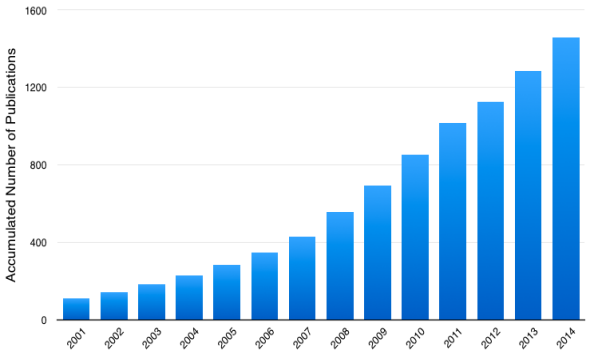


Figure 1: **Growth in SBSE publications**

Since the term SBSE was first coined in 2001 [66], there has been an explosion of activity, creating a body of work that is sufficiently large to support several surveys and analyses of the literature and includes diverse software engineering topics such as requirements, predictive modelling, design, and testing. Figure 1 shows the recent growth[1] in publications on SBSE.

The most closely related work is recent developments in automated repair, which seeks to find patches that fix bugs [89] and Genetic Improvement, which generalises repair, seeking to improve existing programs' functional and non-functional properties, using computational search [86, 112, 113, 138]. Genetic improvement is one example of a recent trend in program improvement, in which existing programs are modified or augmented to improve properties of interest to developers and users [91, 92, 125, 126]. Whereas previous work on genetic programming has tended to build novel code from scratch [79, 87], this more recent trend of program improvement starts from existing systems and seeks to improve them.

## Research Aims and Objectives

My overall aim is to provide automated collaborators; software tools that help developers to adapt and improve existing software systems by recommending changes, backed up by quantitative evidence. Specifically, EPIC will develop and evaluate algorithms and techniques that automatically construct Epi-Collaborators for software transplantation, growing-and-grafting and for uncovering tuneable deep parameters, using evolutionary computation to tailor each to the specific program to be improved. The project will also develop firm scientific foundations, grounded in software testing, verification and fitness landscape analysis, drawing together results from the construction of specific Epi-Collaborators, to provide a general approach to the scientific problem of automated program improvement.

## Overview of Work Packages

EPIC will pursue its goals of transplantation, grow-and-graft and parameter exposition/tuning through three complementary work packages. These three specific targeted work packages will be underpinned by two more foundational work packages that draw their work together and provide cross-cutting scientific foundations in testing, verification, fitness, landscape analysis and hybrids. The work package structure (depicted in Figure 5) has been designed to support seamless extension in the number of specific targeted work packages, thereby supporting incorporation of other Epi-Collaborators.

**WP1 (Specific Targeted Work Package): Automated Transplantation Epi-Collaborator**: This work package will develop techniques for automated software transplantation, which can be used to provide an Epi-Collaborator that oversees the task of automatically transplanting code from one system to another. Automated transplantation will relieve the engineer of a great deal of the laborious and painstaking work involved in this frequent reuse-orientated software development task. Automated transplantation is highly technically challenging, because we need to identify the code to be transplanted and to identify the mappings between name spaces in the two different systems, while ensuring that the transplantation carries over all the desired functionality and avoids any side-effects (regression testing).

[1] Source: SBSE Repository (`http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html`); 2014 is the most recent full year with stable reliable publications data, due to publication time lag. Partial data for 2015 confirms the trend observed, based on historical data for this point in the publication cycle over the past 5 years.

Despite the challenge posed by automated transplantation, my team and I have published proof-of-concept results [18] that demonstrate simple automated transplantation, providing evidence for the feasibility of WP1. These initial results received the distinguished paper award at the International Symposium on Software Testing and Analysis (ISSTA 2015), and won the gold medal for human competitive results at the GECCO 2016 HUMIE awards, demonstrating transplantation's enormous potential.

Many possibilities for automated transplantation are opened up by these initial results, and I want the EPIC project to take it much further: I want advanced transplantation, able to transplant 'near-miss' features (those not implemented in the donor in exactly the required way, but sufficiently close to be useful). I plan to use recent advances in genetic improvement developed by my group [86] to automate such near-miss transplantation. Finally, I want a new kind of transplant-based reuse, by abstracting from multiple transplants to reusable well-documented library functions, using testing as a form of 'disciplined systematic documentation' to explain, justify and document the new library functions.

**WP2 (Specific Targeted Work Package): Grow-and-Graft Feature Extension Epi-Collaborator**: The vision of automated transplantation set out in WP1 will work when the software engineer has available existing systems, with associated legal and technical permissions in place for transplantation. But what happens when this is not the case; how can Epi-Collaboration help when transplantation is not possible?

To address this, WP2 will develop techniques to automatically grow *entirely new* functionality and graft it into existing systems. My work with Bill Langdon and Yue Jia has demonstrated the feasibility of this approach [83]. We automatically grew a bilingual translation system on top of Google translate, and grafted it into Pidgin, a 200,000-line instant messaging system with several million users worldwide, winning the 2014 Symposium on Search Based Software Engineering challenge [67].

Although this proof of concept was exciting and generated a great deal of interest, we are as yet, some way from a complete scientific theory of grow-and-graft software engineering. Our initial work simply used genetic programming to grow a feature which, in isolation, is apparently simply a toy program, but when transplanted into the real-world system becomes a useful additional feature. WP2 will go far beyond this, by exploiting model-based software decompositions (produced by the human) to automatically grow multiple low-level features, and subsequently composing and grafting them into the system, guided by the model. WP2 will also develop hybrids of genetic programming and other techniques such as constraint solving, transformation and analysis (drawn from WP4).

**WP3 (Specific Targeted Work Package): Automated Deep Parameter Epi-Collaborator**: This work package will develop techniques to expose tuneable parameters deeply buried in software systems and previously unknown to the developer of the software. It will then search for values to tune the exposed parameters to optimise system performance. Such parameters need not necessarily simply be scalar values, but could be arbitrary data structures. All that the Epi-Collaborator will require is that the space of possible parameter instantiations is sufficiently well-defined, that it can search for optimal (or near optimal) instantiations of the parameter.

When a software developer creates the initial version of the system, the possible platforms on which it will ultimately be deployed (after many releases) simply cannot be predicted. Therefore, even the most gifted engineer cannot identify and expose all parameters that may ultimately prove to be relevant and important to optimise performance on all possible platforms on which the software will be deployed. However, given a particular platform, an Epi-Collaborator *can* search for additional parameters, for which it can find sufficient evidence of performance improvement, and recommend exposing these parameters. The Epi-Collaborator can then suggest suitable instantiations of parameter choices to maximise performance.

My team and I demonstrated that this parameter exposition approach can automatically identify and expose hidden parameters in widely-used real-world C systems, and can subsequently tune them to optimise both dynamic memory consumption and execution time [141]. WP3 will focus on multiple objectives (one per non-functional property), targeting the multi-objective program improvement vision set out in my keynote paper at the ACM/IEEE Automated Software Engineering conference [68]. However, it will go further by automating experiments that generalise observed improvements to better explain and document the suggestions offered by the Epi-Collaborator.

**WP4 (Foundational Work Package): Testing, Verification, Fitness Functions and Landscapes**: Previous work on genetic improvement has focused on software testing as the primary driver of correctness. Clearly, it will be important to ensure rigorous testing, in situations where we rely on regression testing, for example, to ensure the improved program does not break some existing functionality. However, the project will also use recent results in verification to go beyond what can be achieved merely by software testing alone and to explain and justify Epi-Collaborator suggestions.

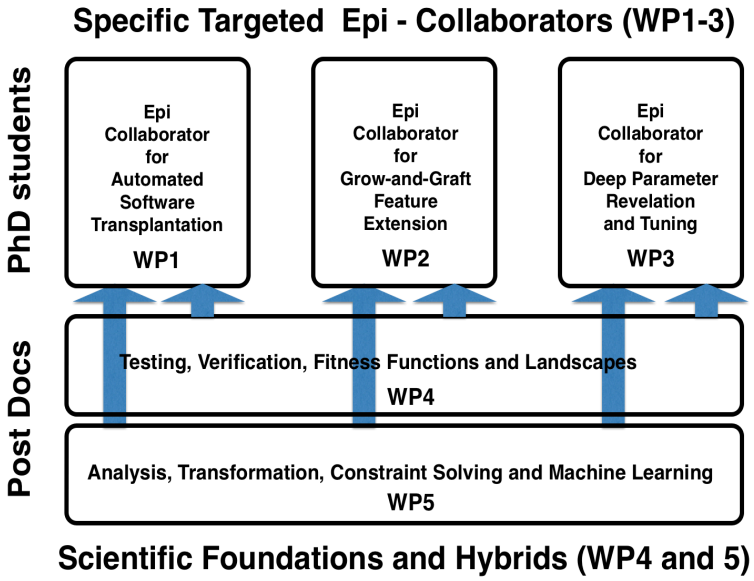


Figure 2: Work packages overview

Any approach to software engineering based on evolutionary computation also relies upon effective fitness functions and candidate solution representations, built on a deep theoretical understanding of the different search landscapes involved.

The EPIC project will provide a sound theoretical foundation on which other researchers, both within EPIC and outside the project, can build. Specifically, WP4 will address four objectives: 1). How can we make use of existing automated test data generation techniques [7] to improve confidence in the verification and validation of genetically improved program variants? 2). How can we deploy recent advances in software verification techniques [40], to give assurances that practically improved program variants are free from certain classes of fault? 3). What is the best fitness function and representation to guide the search towards functionally correct solutions? Closely related to this question, is the concern to find the best representation. 4). What properties characterise the search landscape for different applications; are there commonalities between different specific application domains and how do these affect the choice of genetic operators? These questions will apply across all three specific targeted investigations, each of which will have its own domain-specific fitness functions and representations. From these domain specific choices, we aim to draw out common scientific principles and findings and engineering recommendations. The need to provide high quality guidance for the search process remains an open problem, with existing results highlighting the need for better techniques for guidance [89, 115].

**WP5 (Foundational Work Package): Analysis, Transformation, Constraint Solving and Machine Learning**: While there is strong evidence that evolutionary computation can provide one approach to implement the intelligence required in an Epi-Collaborator, I do not want the project to become dogmatic, merely obsessed with evolutionary computation alone. I believe that other techniques, such as program transformation, constraint solving and program analysis will offer attractive complementary solution approaches that can be combined with evolutionary computation to improve effectiveness and efficiency. These techniques also provide a way to trade the degree of search-space exploration possible against stronger correctness guarantees.

Specifically, WP5 will address four objectives: 1). How can we use analysis, transformation, machine learning and constraint solving to justify, explain and document Epi-Collaborator suggestions? 2). How can we search over semantics-preserving program transformation [20, 35, 43, 61, 134], to produce improvements that are guaranteed to be correct-by-construction; where genetic programming has found attractive solutions, can these be reconstructed by semantics-preserving transformation, thereby ensuring correctness? 3). How can we best exploit and automatically learn from the existing knowledge and insights that derive from program analysis techniques such as program slicing [22, 38, 53], to better guide the search for program improvements and transplantations and to constrain the search space? 4). How can we best incorporate constraint solving techniques [72, 80] to search over the space of program improvements that are correct-by-construction and yet also meet additional human-defined constraints?

### Answers to Some Anticipated Questions

This section anticipates and answers some of the questions for which referees would naturally want answers.

**Won't the code generated become unmaintainable?** The Epi-Collaborators' quantitative evidence, not only justifies and explains improvements to the engineer, but will also provide excellent systematic, consistent and complete documentation. There is evidence that such documentation can *outperform* human-written documentation, because it is reliable and accurate [49]. In this way Epi-Collaborators provide code improvement suggestions *and* detailed, accurate and relevant documentation, relieving the human of much of the (often overlooked) documentation obligations that accompany their change commits.

**How does this differ from previous software engineering assistants?** Existing program assistants automate 'leg work', but leave all the intelligence entirely to the human, whereas EPIC will leave the *decision-making* to the human, but will automate the tedious intellectual activities required for computational search.

**Can evolutionary computation guarantee the correctness of program improvements suggested?** Yes. WP 4 will exploit program verification, while WP5 will develop hybrids with constraint solving, program analysis and transformation that will imbue Epi-Collaborators with the ability to constrain the search to suggestions that are guaranteed to be meaning-preserving (by construction).

**Is the EPIC project simply automating that which engineers already do manually?** Yes. Software engineers routinely seek to transplant existing code, develop new features and to expose and optimise tuneable parameters. EPIC will automate this activity. However, whereas humans cannot fully understand (and thereby optimise) multiple competing objectives, computational search techniques, by contrast, are readily able to 'intelligently' search large multi-objective spaces, while respecting complex constraints.

**Will the Epi-Collaborators impact security and privacy?** Yes. The Epi-Collaborators will help strengthen security by suggesting patches that fix vulnerabilities. Furthermore, the EPIC project will use existing work on measuring information flow and leakage [32] and information theoretic testing [9, 144] as one of the fitness functions to guide the search for improvements that reduce such leakage, thereby improving security and privacy. In this way, Epi-Collaborators provide security and privacy hardening improvements.

**Why these three Epi-Collaborators? What about other Epi-Collaborators?** I have chosen to focus on the three specific Epi-Collaborators of transplantation, grow-and-graft and parameter exposition, because I believe these approaches all show great promise, and are mutually supporting and symbiotic. However, I will also explore other Epi-Collaborators through aligned projects funded by other means, such as self-funding PhD studentships, national funding, and collaborations with visitors to my research group. In my lab, we typically have between two and four visitors at any one time, and several self-funded PhD studentships per year, so this will provide a valuable resource to augment the project. I have carefully designed the project structure to support easy incorporation of such additional Epi-Collaborators.

**What about the intellectual property rights of the transplanted code?** Only the human engineer can know whether he or she has the rights and permissions to perform a transplantation. Nevertheless, the Epi-Collaborator will help considerably by systematically documenting traceability, making the provenance of transplants much easier to understand than with human transplantation. This is another example of the EPIC philosophy of complementary skills; human decides, Epi-Collaborator implements and documents.

**What about requirements and models?** Requirements engineering and model-driven software engineering will both be important for EPIC. For instance, the project will draw on the latest results in elicitation and understanding of non-functional requirements, while models will be used to help understand system decomposition, guiding the Epi-Collaborators towards manageable sub-components of the model.

**Isn't the UK leaving the EU: what happens about Brexit?** This proposal stands on the basis of the scientific excellence evidenced above. The team I lead at UCL is deeply embedded in a network of leading European scientific institutions. This collaboration has always been and will remain critical to success and it will be sustained. The U.K. Government is committed to ensuring the engagement of UK scientists in European projects on an ongoing basis.


Figure 3: **The Global Spread and Geographic Intensity of SBSE Authorship**

## Part B2a: State of the Art and Objectives

### State of the Art

The EPIC project lies primarily within the field of Search Based Software Engineering (SBSE) [48, 66, 69], particularly recent developments in Genetic Improvement and related techniques that seek to automatically improve existing software systems [86, 91, 92, 112, 125, 126, 138]. This section summarises the state-of-the-art in SBSE and Genetic Improvement.

**SBSE** uses computational search techniques, and in particular evolutionary computation [59], to tackle demanding optimisation problems in software engineering, characterised by multiple conflicting and competing objectives. There has been an explosion of activity, as illustrated by the global spread of SBSE research, depicted[2] in Figure 3. This explosion in SBSE work has created a body of research that is now sufficiently large that it has many surveys and analyses. These surveys cover specific subareas of SBSE research targeting a wide range of software engineering activities, domains and problems, including software project management and estimation [3, 44], requirements [45], design [62, 117], non-functional properties [4] and testing [5, 65, 99, 142].

SBSE has also found its way into industrial practice. One of the earliest industrial examples of the application of SBSE was the Daimler evolutionary testing system [137]. More recently, Microsoft used SBSE to handle floating point computation in its Pex software testing tool [27, 82], while Google incorporated SBSE regression test optimisation into its test process [145]. NASA [36], Motorola [14] and Ericsson [146] have also experimented with SBSE for requirements analysis and optimisation.

There are many SBSE tools that build on the scientific foundations of SBSE and that support SBSE applications right across the range of software engineering activities, from release planning [105], through design [102] to testing [6, 47, 76, 81, 131] re-modularisation and refactoring [102, 103] and repair [90]. Tools such as EvoSuite [47] and AUSTIN [81] for testing, GenProg [90] for repair and Bunch [102] for re-modularisation are well-known and widely used. However, none of these tools offers the kind of collaborative assistance envisaged by the EPIC project. The closest related work to the EPIC project lies in the area of Genetic Improvement.

[2]Source: SBSE repository: `crestweb.cs.ucl.ac.uk/resources/sbse_repository/`

**Genetic Improvement** is a recently emerging trend within SBSE that has had notable recent success. Genetic Improvement uses computational search to improve existing software systems' functional and non-functional properties. Recent results have demonstrated that genetic improvement can repair broken functionality [12, 89], dramatically scale-up benchmarks [112, 138], and also real-world systems, such as a DNA sequence analysis tool of 50,000 line of C++ [86], graphic shaders [126] and 2D Stereo and 3D medical imaging systems [84]. The related loop perforation approach has also been demonstrated to speed up several systems [125], while other program improvement-based approaches have reduced energy [26, 91, 92, 124] and dynamic memory [141] consumption. However, none of these previous approaches provide evidence to justify and explain the improvements, nor do any offer correct-by-construction improvements as proposed for the EPIC project and few cater for multiple objectives [139, 141].

### The Multiplicity Problem

Software engineering has always been an exceptionally demanding activity, stretching human intellect and ability to its very limits. Unfortunately, the demands of the emerging 'multiplicity paradigm' [29], require the ability to optimise multiple competing objectives: there are multiple different platforms, each with its own characteristics and peculiarities, and for each of these different platforms there may be multiple different user communities, each with their own set of requirements and needs.

Yet more challenging are the multiple sets of operational (aka non-functional) requirements that need to be met in emerging software engineering paradigms, such as mobile computing and Internet of Things. Software systems have always had to cater for speed and space objectives, to which compiler technology has become accustomed and well-adapted to tackle. However, in the newer multiplicity paradigm, software engineers need to simultaneously optimise for many more competing and conflicting objectives, including energy consumption, dynamic memory consumption and communication latency. Not only are these objectives in conflict but, equally importantly, the impact of changes made by the software engineer on each of these objectives is poorly understood. Further incremental improvements in compiler technology are insufficient to handle this multiplicity problem.

Faced with these problems of multiple platforms, multiple objectives and multiple sets of requirements, software engineers have developed a number of coping technologies, such as model-driven approaches to software engineering [34, 46, 114] (for which SBSE is also beneficial [62]). Models abstract away from details, allowing multiple instantiations. This is important in managing problems of scale, but it does not directly tackle the problem of selecting from the enormous space of possible instantiations, many of which will turn out to be highly suboptimal, while only a few (per platform of interest) will prove to be well-optimised for the multiple objectives required.

Traditionally, few software engineers would even *consider* an approach that attempts to search this space; because of the limits of human intellectual ability, no human-based methodology would be feasible. This is why the EPIC approach is a radical departure from tradition: it will develop techniques that automatically search this enormous space, thereby helping the human to navigate the space of possible software system instances, using Epi-Collaborators that augment human engineering abilities with systematically documented, explained and justified improvement suggestions.

Current software development practices risk retreating into an ever more conservative one-size-fits-all approach. Without an approach like EPIC, we will continue to generate increasingly less optimal software systems, ill-adapted to their platforms, as the search space inevitably and inexorably grows. More labour-intensive software development is not only costly and time-consuming, but also wastes natural resources, such as energy. Indeed, an increasing proportion of global energy consumption derives from computation [127, 136]. If we cannot find a step change in our software development techniques, we will increasingly squander precious resources, as software systems become ever less optimal in the face of rising multiplicity.

### The EPIC Philosophy

The thesis that underlies my approach to the EPIC project is that humans and machines have complementary skills and that, as a result, we need software development methods that not only respect this complementarity, but actively exploit it to radically augment existing human skills with 'intelligent' computational search. EPIC represents a fundamental shift in the demarcation between human and machine,

significantly augmenting human cognitive ability by dramatically extending software engineers' ability to navigate complex multi-objective system improvement spaces.

Faced with the enormous problems of multiplicity, I believe it is only natural to deploy computational search techniques, such as evolutionary computation, to navigate this space. My team and I at UCL have recently demonstrated that SBSE techniques are capable of finding useful code fragments in this enormous 'multiplicity space' that optimise for speed, energy consumption, and dynamic memory [26, 141]. We have also demonstrated that computational search can be used to reconstruct core functionality, and to grow-and-graft it from one system to another [67, 85]. Finally, using a combination of program analysis and genetic programming [18], we have been able to identify, extract and transplant functionality from one (donor) system to an entirely unrelated (host) system.

These initial proof of concept results demonstrate SBSE's potential to contribute to the software development process in ways that were previously the preserve of humans. However, I believe we have merely scratched the surface of what is possible. Our initial results are important, because they demonstrate feasibility of the approach set out in this proposal, but EPIC will fully realise this vision.

**Empirical Results that Support the EPIC Philosophy**

Notwithstanding our initial proof-of-concept results, there remains a high degree of challenge in the project. Fortunately, there are also recent empirical results that support the central thesis of EPIC, which indicate that Epi-Collaborators will find useful improvements. For instance, we know from the findings of Gabel and Su [50] that a surprisingly large amount of code has to be written before the software engineer will have produced something that does not already exist elsewhere in the software repository. All code fragments of fewer than seven lines of code can be found somewhere else in SourceForge [50], so we know that any improvement that involves fewer than seven lines of code could potentially be found by an Epi-Collaborator. More recently, my own group presented an extensive empirical study of real-world code commits that indicated that approximately 43% of code committed to repositories by software engineers can be constructed by simple grafts of existing code in the software repository [17] (see Figure 4), an observation that has further stimulated interest in Genetic Improvement and related research.

These findings underscore the feasibility of searching for existing fragments of code that can be partly rewritten and recomposed to improve software systems. However, in the EPIC proposal, EPIC will go beyond merely recombining fragments of code. I plan to use genetic programming, guided and constrained by automated software testing, verification, constraint solving, analysis and transformation to evolve improved programs, drawing upon existing fragments, but not relying solely on their re-combination.

**Research Aims and Objectives**

The overall aim of the project is to provide automated collaborators; software tools that help developers to adapt and improve existing software systems, by recommending changes, backed up by quantitative evidence, automatically generated by the collaborator.

Specifically, EPIC will develop and evaluate algorithms and techniques that automatically construct Epi-Collaborators that can transplant features from an existing system, grow entirely new features and graft them into a system, and that can expose and tune previously hidden parameters, all with respect to multiple objectives and providing evidence and justifications for the improvements suggested. These Epi-Collaborators are not only complementary to the engineers' skills, but they are also mutually complimentary to one another; an engineer can combine Epi-Collaborator results, for example exposing and tuning parameters in transplanted code.

The EPIC project will also develop firm scientific foundations, grounded in software testing, verification and fitness landscape analysis, drawing together results from the construction of specific Epi-Collaborators, to provide a general approach to the scientific problem of automated program improvement.

**Scientific Challenges**

Epi-Collaborators will need to demonstrate a level of intelligence significantly beyond the one provided by current software engineering automated support. This poses many scientific challenges, but fortunately the time is ripe for the EPIC approach, because the techniques on which it will draw (testing, verification, constraint satisfaction, machine learning, program analysis and transformation) have all recently matured to the point at which each can be brought to bear to dramatically extend the power of computational

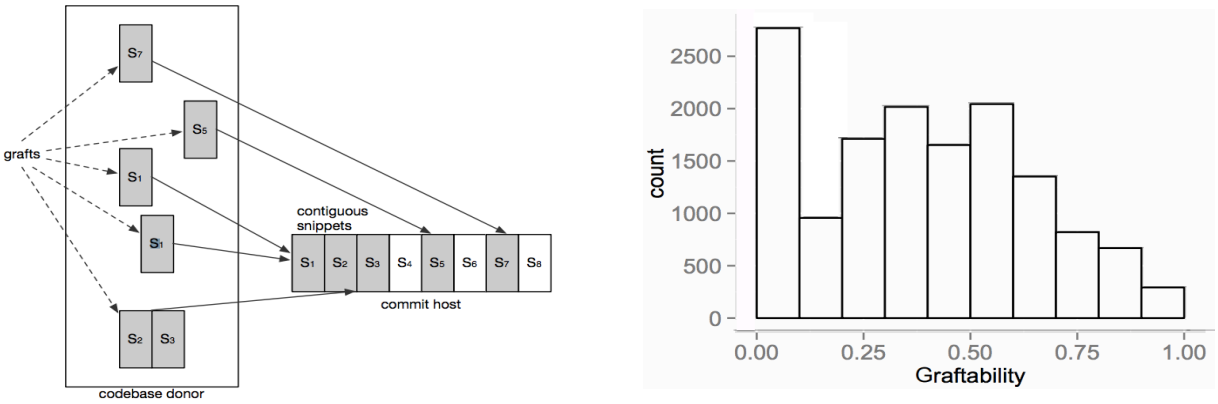


Figure 4: **The Surprising Graftability of Human-Generated Software Commits** [17]. We investigated the degree to which commits, made by engineers, to the software repositories of 12 different software systems could have been composed using fragments of existing code already in the system updated by the commit. The left hand side of the figure depicts the experimental procedure we followed. We consider the existing code base of the system, as the source of potential grafts; a set of snippets $S_1 \ldots S_n$. We then measure the percentage of the commit made by the human engineer (the commit host) that could have been formed using only these snippets. On the right hand side, we present a histogram of the results, in which the horizontal axis denotes graftability (the percentage of a commit that can be composed entirely from existing snippets, in buckets of 10%). The vertical axis of the histogram reports the number of commits found to contain $x$% of snippets, where $x$ is the location of the histogram bar on the horizontal axis. In total, we considered 15,723 different commits, finding a surprisingly high level of graftability (43%), providing compelling feasibility evidence for a search based approach to program improvement.

search. In this section, I briefly outline four such challenges, while the next section explains the ways I propose to draw on these techniques to support computational search.

**Automated experimentation**: How to automate experimentation to move from specific code interventions to general characterisations of the intervention space, thereby providing more valuable feedback and robust interventions. Epi-Collaborators need to be able to generalise and abstract from specific observations, for example, parameter values that improve performance.

**Near-miss transplantation**: How to fully automate transplantation of near-miss donor code that does not quite meet the required functionality of the host. This will radically increase the pool of potential transplants from which Epi-Collaborators can extract donor code, and will also extend the types of features that can be transplanted, since we can pick features that are almost-but-not-quite fit for a given purpose, and improve them to make them fit-for-purpose.

**Correctness verification**: How to target verification, constraint solving, analysis and transformation techniques to provide incontrovertible evidence of the efficacy of transplants and code improvements.

**Scalability**: How to find efficient computational search techniques based, for example, on theoretical and experimental understanding of search landscape characteristics, so that Epi-Collaboration can scale to real-world software engineering problems.

All of these challenges will be tackled by the project, as explained in the next section on the methodology to be adopted by the EPIC project.

## Part B2b: Methodology

The project structure is illustrated in Figure 5. The project consists of five work packages. The overall project structure has two 'horizontal' (cross cutting) foundational work packages that will be undertaken by post doctoral researchers, and three vertical work packages that target specific Epi-Collaborators (and which will be tackled by PhD students).

Work packages WP1, WP2 and WP3 are specific targeted work packages, each of which will implement a particular approach to Epi-Collaboration. They were described in the synopsis in Part B1, together with an overview of work packages WP4 and WP5. This section includes the descriptions of work packages WP1, WP2 and WP3 from Part B1 of this proposal for ease of reference. It then gives a more in-depth account of the technical and scientific foundations of the project, that will be tackled by the foundational work packages WP4 and WP5, and the cross-cutting software engineering aspects that will apply throughout the project.

**WP1 (Specific Targeted Work Package): Automated Transplantation Epi-Collaborator**: This work package will develop techniques for automated software transplantation, which can be used to provide an Epi-Collaborator that oversees the task of automatically transplanting code from one system to another. Automated transplantation will relieve the engineer of a great deal of the laborious and painstaking work involved in this frequent reuse-orientated software development task. Automated transplantation is highly technically challenging, because we need to identify the code to be transplanted and to identify the mappings between name spaces in the two different systems, while ensuring that the transplantation carries over all the desired functionality and avoids any side-effects (regression testing).

Despite the challenge posed by automated transplantation, my team and I have published proof-of-concept results [18] that demonstrate simple automated transplantation, providing evidence for the feasibility of WP1. These initial results received the distinguished paper award at the International Symposium on Software Testing and Analysis (ISSTA 2015), and won the gold medal for human competitive results at the GECCO 2016 HUMIE awards, demonstrating transplantation's enormous potential.

Many possibilities for automated transplantation are opened up by these initial results, and I want the EPIC project to take it much further: I want advanced transplantation, able to transplant 'near-miss' features (those not implemented in the donor in exactly the required way, but sufficiently close to be useful). I plan to use recent advances in genetic improvement developed by my group [86] to automate such near-miss transplantation. Finally, I want a new kind of transplant-based reuse, by abstracting from multiple transplants to reusable well-documented library functions, using testing as a form of 'disciplined systematic documentation' to explain, justify and document the new library functions.

**WP2 (Specific Targeted Work Package): Grow-and-Graft Feature Extension Epi-Collaborator**: The vision of automated transplantation set out in WP1 will work when the software engineer has available existing systems, with associated legal and technical permissions in place for transplantation. But what happens when this is not the case; how can Epi-Collaboration help when transplantation is not possible?

To address this, WP2 will develop techniques to automatically grow *entirely new* functionality and graft it into existing systems. My work with Bill Langdon and Yue Jia has demonstrated the feasibility of this approach [83]. We automatically grew a bilingual translation system on top of Google translate, and grafted it into Pidgin, a 200,000-line instant messaging system with several million users worldwide, winning the 2014 Symposium on Search Based Software Engineering challenge [67].

Although this proof of concept was exciting and generated a great deal of interest, we are as yet, some way from a complete scientific theory of grow-and-graft software engineering. Our initial work simply used genetic programming to grow a feature which, in isolation, is apparently simply a toy program, but when transplanted into the real-world system becomes a useful additional feature. WP2 will go far beyond this, by exploiting model-based software decompositions (produced by the human) to automatically grow multiple low-level features, and subsequently composing and grafting them into the system, guided by the model. WP2 will also develop hybrids of genetic programming and other techniques such as constraint solving, transformation and analysis (drawn from WP4).

**WP3 (Specific Targeted Work Package): Automated Deep Parameter Epi-Collaborator**: This work package will develop techniques to expose tuneable parameters deeply buried in software systems and pre-

viously unknown to the developer of the software. It will then search for values to tune the exposed parameters to optimise system performance. Such parameters need not necessarily simply be scalar values, but could be arbitrary data structures. All that the Epi-Collaborator will require is that the space of possible parameter instantiations is sufficiently well-defined, that it can search for optimal (or near optimal) instantiations of the parameter.

When a software developer creates the initial version of the system, the possible platforms on which it will ultimately be deployed (after many releases) simply cannot be predicted. Therefore, even the most gifted engineer cannot identify and expose all parameters that may ultimately prove to be relevant and important to optimise performance on all possible platforms on which the software will be deployed. However, given a particular platform, an Epi-Collaborator *can* search for additional parameters, for which it can find sufficient evidence of performance improvement, and recommend exposing these parameters. The Epi-Collaborator can then suggest suitable instantiations of parameter choices to maximise performance.

My team and I demonstrated that this parameter exposition approach can automatically identify and expose hidden parameters in widely-used real-world C systems, and can subsequently tune them to optimise both dynamic memory consumption and execution time [141]. WP3 will focus on multiple objectives (one per non-functional property), targeting the multi-objective program improvement vision set out in my keynote paper at the ACM/IEEE Automated Software Engineering conference [68]. However, it will go further by automating experiments that generalise observed improvements to better explain and document the suggestions offered by the Epi-Collaborator.

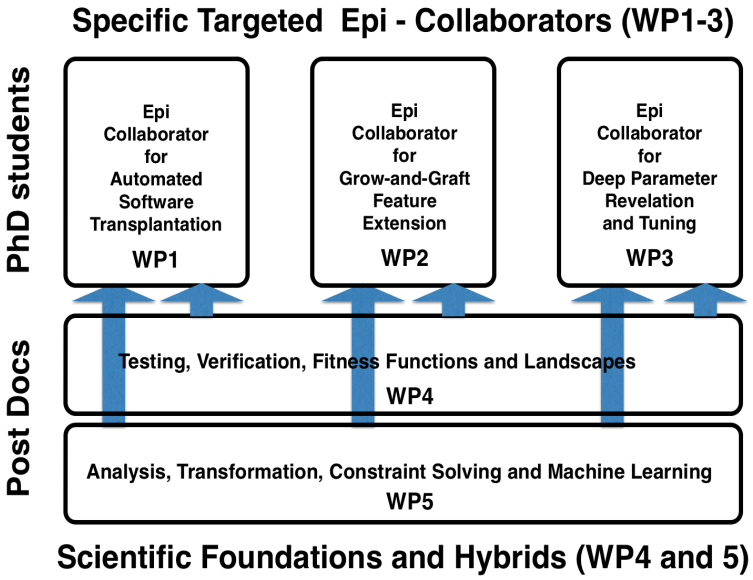


Figure 5: Work packages overview

**WP4 (Foundational Work Package): Testing, Verification, Fitness Functions and Landscapes**

WP4 will develop algorithms and methods for automated software testing and verification of genetically improved programs. Here I explain the ways in which we will advance testing techniques, and use verification, fitness and landscape analysis to develop the theory and practice of genetic improvement.

WP4 will be applied across all of the different Epi-Collaboration improvement techniques developed in the 'vertical' work packages (WP1, WP2 and WP3), each of which concerns a specific Epi-Collaborator.

**WP4.1, Testing**: Testing will naturally be a core approach that EPIC will use to guide the Epi-Collaborators towards solutions that remain faithful to the functionality of the original program they seek to improve. Testing will also be core to the EPIC project because it denotes one route towards providing the evidence Epi-Collaborators will use to justify and explain their suggestions. In WP4.1, EPIC will advance the field of automated testing, while in WP4.2 it will exploit recent advances in verification.

**Leveraging Automated Testing**: The quality of test cases will clearly have a critical impact on the quality of genetically improved code. This is partly because testing will be used to guide the fitness function, but also because verification techniques may be inapplicable in some cases (for example, due to limited scaleability of the underlying verification technology). Fortunately, there have recently been great strides in automated testing [7]. Search based software testing [65, 99], dynamic symbolic execution [27], and their hybrids [13, 74] can generate test inputs that achieve good coverage of code components.

There are several high-quality tools for automated test data generation available [47, 81, 130]. While there remain open problems and issues with automated testing, existing techniques provide a reasonable

scientific and engineering basis on which to build, both better automated genetic improvement, and also better automated test data generation.

**Developing Novel Automated Test Generation**: Existing automated test technology is relatively mature when it comes to structural coverage, but there is less maturity in the area of testing non-functional properties [4, 65]. The deployment of automated test technology to the problem of quality assurance and guidance for genetic improvement raises a number of interesting scientific and engineering problems. For example, 'how can we best target test data generation at non-functional properties, such as energy consumption and throughput?' [65]. Search based software testing is a natural starting point here, since any property that can be measured could, in theory, also be formulated as a fitness function to guide the search for test data. EPIC also poses interesting challenges to traditional test data generation: because we are trying to *improve* the system under test, some of the existing test cases may become inapplicable. To address this problem we shall not only have to generate test cases to cover new code, but we shall also regenerate test cases to cover new requirements, not anticipated when the original test suite was constructed. To address this problem, we shall start by considering test case augmentation and regeneration [122, 143].

**Co-evolutionary optimisation**: EPIC will exploit co-evolutionary optimisation [1, 11]. In a co-evolutionary optimisation problem, two or more populations evolve simultaneously, such that the fitness function of each population is defined by the current members of the other. In the literature on search based software engineering, both co-operative [119] and competitive [1, 11] co-evolution have previously been studied. The EPIC project will use competitive co-evolution to generate improved programs and test cases that are better at revealing faults in these programs. Our goal will be to stimulate a 'predator-prey arms race' between the test cases and the improved programs they test. In this competitive co-evolutionary model, the improved programs play the role of prey, while the test cases play the role of predators. The better adapted the tests are at revealing faults, the better they play their role as predators. The aim is that this, in turn, stimulates the improved programs to become better at passing even the most demanding co-evolved test cases: test cases that have been evolved specifically to be predators, which are well-adapted to exploit weaknesses in their prey.

**WP4.2, Verification**: In WP4.2 EPIC will draw on recent advances in software verification, using verification techniques to check for correctness, and to provide the evidence to support the correctness claims of Epi-Collaborators, where correctness is found to hold.

**Maintaining Correctness**: I will collaborate with Daniel Kroening and his group at the University of Oxford, and verification researchers at UCL to incorporate and extend recent developments such as regression verification [54] to provide stronger correctness guarantees for improved programs, fragments and transplants produced by the Epi-Collaborators.

Our notion of correctness, inspired by regression verification, will be that the improved system should be 'as correct as' the original, thereby augmenting and potentially strengthening the assurances over traditional genetic improvement, which relies solely on test adequacy alone. My team and I have already held two full day informal workshops with Daniel Kroening and his group at Oxford. He has kindly agreed to collaborate with us on this stream of the EPIC project.

I will also, naturally, draw on expertise from our own UCL departmental colleagues working on verification: Peter O'Hearn (with whom I have already begun collaboration) and the UCL Programming Principles, Logic and Verification group (http://pplv.cs.ucl.ac.uk). This stream of investigation will allow us to investigate whether we can provide a verification-based guarantee that the genetically improved program is, at least, as correct as the program from which it has been improved.

**Correct-by-Construction Genetic Improvement**: While the deployment of 'as correct as' guarantees will give us a stronger guarantee than pure regression testing alone, we can go further by seeking to produce genetically improved programs that are correct-by-construction in some situations. We shall explore both constraint-based genetic improvement and transformation-based genetic improvement.

**Correctness as an Objective**: A natural question arises: if we are successful in finding techniques that are correct (by construction), then why would we wish to seek any other kind of genetic improvement?

There are two answers to this question, one scientific and the other based on engineering practicalities. 1. (Scientific issue): Transformation sequences and constraints may overly constrain the search space, so that globally optimal (and correct) solutions may not be found, as has been found with other other highly-constrained search problems [55]; 2. (Engineering issue): There may be situations in which correctness-by-construction genetic improvement simply does not scale sufficiently to apply to larger systems. In order to tackle these issues, we shall explore techniques that combine correctness-by-construction genetic improvement with other forms of genetic improvement. This allows us to consider a search space in which correctness itself is an objective, rather than a boolean property of the system. With such a hybrid approach, we hope to produce improved versions of the program that are sufficiently exciting and interesting to the programmer that he or she will be prepared to discharge any remaining correctness obligations, left open by the partial use of correctness-by-construction improvement techniques.

**WP4.3: Fitness Functions and Landscapes** WP4.3 will draw out and develop theoretical and practical aspects of representation, operator and fitness function design, investigating the impact of these choices on solution quality and algorithm performance. This work will cut across all of the three specific targeted work packages and also draw on and feed into other research work on genetic programming and search based software engineering, both within my lab, CREST, and within the wider community.

We will use theoretical approaches, such as landscape analysis [31, 140] and algorithmic analysis [70] to more fundamentally understand the structure and properties of the search landscapes imbued by the software engineering problems we will tackle. This will support a deeper understanding of algorithm performance, offering theoretical bounds on performance and solution quality. This theoretical analysis will not be performed in isolation, but will be conducted in partnership with the empirical and experimental analysis, using empirical results to test theoretical assumptions, and experimental analysis to investigate the practical implications of theoretical bounds.

**Linear Genetic Programming:** Most existing genetic improvement work uses a variable-length linear representation of program improvements. However, other Genetic Programming (GP) applications use syntax tree-based representations [79]. Linear GP [16] approaches continue to prove important and applicable, while Nordin's [108] work has been incorporated into the commercial GP tool Discipulus. We will experiment with different representations, in order to better understand their impact. This issue will naturally also arise in work package WP4.3, but will cut across all three of the specific targeted work packages, since each may find that a different representation is suitable for their problem(s).

**Cartesian Genetic Programming:** Another potentially attractive approach for genetic improvement lies in the use of Cartesian Genetic Programming [101]. This has proved to be one of the most successful variants of GP, with more than 220 papers being published. Using Cartesian Genetic Programming, the tree is replaced by a grid of computing nodes in which both their contents and connectivity is evolved. Cartesian Genetic Programming has been widely applied [100] and extended [135]. However, it has not previously been used in genetic improvement, but its examination for this field is long overdue: existing genetic improvement approaches fail to make anything but the most basic distinctions between phenotype and genotype. More work is clearly required to understand what possibilities emerge when this link is stretched further. The previous work on Cartesian Genetic Programming has shown that moving to a space that facilitates evolution of fit individuals can dramatically improve performance and solution quality. There is no scientific reason to suppose that similar performance improvements will not also apply to genetic improvement work.

**Grammatical Evolution:** Another highly successful approach is Grammatical Evolution [111]. Using Grammatical Evolution, a linear genome is used to navigate a path through a user-supplied BNF grammar. At each branch point in the grammar, the next element in the genome deterministically decides which production rule to apply. When terminals in the grammar are reached they are output and become the GP individual's phenotype.

This form of genetic programming may allow us to include constraints on the search space, which may be naturally formulated in terms of grammatical production rules. In this way, we may be able to incorporate correctness and other concerns as constraints on the search space.

**Genetic Operators and Fitness:** WP4.3 will also consider the interplay between the representation, the genetic operators and the fitness function. Naturally, for each of the specific targeted projects, there will be different fitness functions, since these are largely domain specific. WP4.3 will draw out common scientific and engineering principles, on which other researchers may build. Such generalisation from specifics will necessarily involve the exploitation and development of recent results concerning the use of genetic programming operators. For example, the 'standard' tree crossover [79] can create offspring that do not lie 'between' the crossover's chosen parents. Moraglio suggested a geometric interpretation which he and co-authors have used to devise new types of genetic programming leading to Geometric Semantic Genetic Programming [104].

**WP5 (Foundational Work Package): Analysis, Transformation, Constraint Solving and Machine Learning**: I have spent much of my career demonstrating that evolutionary computing has a lot to offer software engineers, but I do not believe that it is the only answer to all problems. Indeed, I believe hybridisation with more traditional programming language techniques, such as analysis and transformation, is essential to make evolutionary search practical, scalable and applicable to real-world software engineering systems. I also think other important techniques, associated with the broad area of Artificial Intelligence (AI) are important to complement and augment search based techniques [60]. WP5 will develop hybridised algorithms that exploit these traditional software engineering and AI techniques, combining them with evolutionary computing. This will not only be scientifically important for the EPIC project, but through this hybridisation work, EPIC will additionally play its role in bringing together computer science communities that would not otherwise often collaborate and communicate with one another.

**WP5.1: Automated Experimentation for Evidence** EPIC's Epi-Collaborators will offer suggestions for improvements to software systems to the engineer, backing these up with evidence from automated experiments, to justify and explain the suggestions made. Ultimately, these will be developed to incorporate machine learning techniques, guided by engineer-feedback to specifically tailor the Epi-Collaborators to the engineers they serve. WP5.1 will investigate two kinds of automated experimental evidence provision:-

**Experiments to Generalise From Specific Instances**: Epi-Collaborators will automate experiments that generalise specific improvements they find, characterising general improvement properties, aiming to better document, explain and justify the Epi-Collaborator's suggestions. For example, suppose the Epi-Collaborator finds that `WidgetSize`, has a value 17 that optimises the execution time. The Epi-Collaborator will automatically run experiments to find the underlying relationship, using genetic programming to evolve an equation that captures the relationship between `WidgetSize` and execution time.

We will use parsimony as part of the overall fitness function, in order to aid readability of the evolved equation, and only pass the result to the engineer if it satisfies readability constraints. In this way, an Epi-Collaborator might recommend a value 17, but it would also give the engineer an equation that explains the relationship, thereby better explaining why 17 is the optimal choice recommended.

We will develop other automated experiments to generalise by generating assertions from test cases [42]. For example, starting from the Epi-Collaborator's observation that `WidgetSize`=17 at line 21,890 is found to give best execution time, the Epi-Collaborator will use search based software testing [65] to generate a test suite that traverses this portion of the code, using test cases to guide assertion inference.

Once again, we will use readability constraints to avoid bombarding the engineer with assertions. The result of such experiments will be reported in the form: "When `WidgetType` = `NormalDisplay` and `Device` = `iphone`, then $15 \leq$ `WidgetSize` $\leq 17$ will give the best observed execution time".

Clearly, much can and needs to be done in order to make this feedback as readable as possible. Initially, we will simply prioritise the assertions and equations auto-generated through experimentation, according to 'obvious' metrics (length, number of free variables and sub-terms etc.). However, we can then use engineer feedback to tune the prioritisation, thereby inferring new (and inherently well-tailored engineer-specific) readability metrics to better guide future prioritisation.

Ultimately, I aim to experiment with reinforcement-based machine learning to specifically tailor the experimental evidence to the engineers to whom it is offered. For this goal, EPIC will benefit from the excellent machine learning group at UCL, with its expertise in Support Vector Machines, Monte Carlo Tree Search and Deep Learning.

**Experiments to Enhance Test-Based Justification**: One of the advantages of all three Epi-Collaborators proposed in work packages WP1 – WP3 is that the techniques will automatically be aware of the specific locations to be affected by transplantation, grafting and parameter tuning. This knowledge will facilitate specific targeted testing, to generate a test suite that exercises and repeatedly covers the areas of code affected by the Epi-Collaborators' suggestions, and those revealed by dependence analysis to be transitively affected.

Traditionally, automated test data generation is profoundly inhibited by the so-called 'oracle problem' [19, 128]. That is, although we may automate the generation of test inputs, how can we tell what the desired output should be? Due to this oracle problem, traditional test data generation is only able to generate test inputs, but cannot generate the corresponding expected output, with the result that testing is only partially automated; humans still have to provide oracle information. Fortunately, EPIC does not follow the traditional test data generation scenario, because it benefits from much deeper knowledge of the changes made to the system under test. Therefore, the oracle problem will be significantly reduced for Epi-Collaborators. For example, a critical justification Epi-Collaborators will need to provide to the engineer will be compelling regression testing [41, 120, 142]. In this situation, not only do we know exactly which parts of the code have been suggested for change by the Epi-Collaborator, but we will have the original program to act as a regression test oracle, thereby facilitating complete automation.

**WP5.2: Searching Over Semantics-Preserving Transformations**: EPIC will use automated search based transformations, which are meaning-preserving by design, in order to find sequences of transformations that improve the non-functional properties of programs. Since each transformation sequence is meaning preserving, the overall application of the sequence of transformations will also be meaning preserving. This strand of our work will draw on the long heritage of research in search based software transformation and re-factoring [25, 109, 121].

WP5.2 will also use transformation, but to provide explanations and justifications of the improvements suggested. Where a transformation sequence can be found, consisting entirely of meaning preserving transformations, that ultimately transforms the initial program to an improved version, this transformation sequence will provide an excellent justification and documentation of the change suggested.

It also forms a means of communication between engineer and Epi-Collaborator, because the engineer can refuse certain transformations as undesirable, where, for example the engineer believes they may harm ongoing maintenance or because he or she simply does not understand them. Because EPIC's fundamental underlying approach is search based, this need not be a problem; the Epi-Collaborator can search for alternative transformation sequences to 'fill in the gaps' left by such engineer refusals. Such refusal sets also denote a further valuable feedback mechanism, providing a guidance for machine learning and tailoring of the Epi-Collaborator to each engineer's tastes and cognitive preferences.

**WP5.3: Incorporating Constraint Satisfaction Techniques**: EPIC will use constraint solving techniques to find fragments of code that improve existing programs. The constraints in this case capture the correctness obligations for the code we graft into the existing program to improve it. This approach is inspired by recent work on constraint-based automated repair [107]. Essentially, we seek to generalise constraint-based automated repair to constraint-based automated improvement; repair being a special case of improvement, in which the bugfix is the improvement.

**WP5.4: Exploiting Knowledge and Insights from Program Analysis**: EPIC will also use program analysis, and in particular dependence analysis [15, 23, 37, 73, 118], to document and explain its suggestions. This will be particularly important in automated transplantation, for which dependence analysis can explain the provenance of the code transplanted, and the nature of its dependence upon the core functionality that the software engineer seeks to transplant. Such dependence analysis can also be used to automatically maintain traceability links, another important software engineering concern, as explained in the next section on relationships to the requirements and other software engineering communities.

**Developing and Enhancing Relationships to Other Research Areas**

In the foundational work packages, I have explained the way in which EPIC will draw on and contribute to research in constraint solving, program transformation, program analysis, machine learning and verification. However, cutting across the three specific targeted work packages there will be strong links with the software engineering research areas of Model Driven Software Engineering, Requirements Engineering, Recommender Systems, Computer Supported Cooperative Work, and Testing. Throughout the project EPIC will also draw on and contribute to research in Evolutionary Computing.

Through workshops, sabbatical visits, and tutorials, facilitated by the project, I plan to develop and strengthen the links between the software engineering community and the evolutionary computing communities, and between EPIC and these other areas of software engineering. For these workshops I will use the highly successful CREST Open Workshop (COWs) Programme[3], inviting leaders in these communities as keynotes and invited speakers. Since 2009, my centre, CREST, has hosted 46 such COWs at UCL (449 research talks so far), with 1,624 registrations from 252 institutions, spread over 44 countries.

**Relationships with Model Driven Software Engineering**: Model based software engineering is increasingly important to cope with both the scale of software systems and the multiplicity problem [46]. Models of software systems are expressed at a higher level of abstraction than code, thereby helping software engineers to cope with scale. This will also be useful to the EPIC project; such models (designed by the human) will help the machine, directing it to smaller components that can be grown and grafted into systems or transplanted from elsewhere. In this way, models will form a convenient division of responsibilities between human and machine.

Model-based testing will also offer another route to the generation of the test cases that guide and constrain Epi-Collaborators [21, 51, 129], while model-based slicing [8, 10] will be used to extract sub-models, targeting Epi-Collabroators at particularly important aspects. We recently demonstrated techniques for inferring test models [132], winning the 2016 SBSE challenge [147], and will leverage these techniques to augment the testing used to explain and justify suggestions from Epi-Collaborators.

Models have also been used to cope with multiplicity, by capturing variability, and such models would also be very useful to the EPIC project, allowing its automated techniques to focus on the smaller components of Software Product Line architectures, to automatically extend them for specific instances, and to merge branches to tackle branchmania [24]. I outlined this vision of 'Search Based Software Engineering for Software Product Lines' in my keynote at the Software Product Line Conference in 2014 [62]. EPIC will build on and develop this interplay between Model Driven Engineering and Search Based Software Engineering, supporting and nurturing workshops and collaborations at the intersection.

**Relationships with Requirements Engineering**: For any large-scale software engineering activity to be successful, careful attention to the elicitation and management of requirements will be essential [30, 133]. Requirements Engineering is thus rightly a field of great importance for the software engineering community, and one to which SBSE techniques have proved effective [45, 57, 71, 78].

In order to be successful, it is essential to maintain traceability links between requirements, models, code and the test cases [33, 56]. This is a demanding problem for engineers, requiring attention to detail and careful updating and consistency checking. One of the strengths of EPIC is the way in which it will self-document, helping the engineer to maintain, justify, explain and document the connection between code improvements, the requirements they satisfy, and the test cases that witness this satisfaction.

I plan for EPIC to make contributions to requirements engineering, focusing on traceability and optimisation of requirements choices. EPIC will allow engineers to more quickly prototype new features. This will be important in domains such as mobile computing that undergo rapid release cycles [96, 98] and feature migration [123]. EPIC will investigate the extent to which Epi-Collaborators support exploratory requirements elicitation; allowing the developer and requirements engineer to work together to explore and rapidly prototype potential feature extensions, thereby investigating speculative requirements. However, I also plan to learn from and incorporate developments from the requirements engineering community, particularly in the elicitation and understanding of non-functional requirements.

[3] http://crest.cs.ucl.ac.uk/cow/

**Relationships with Recommender Systems and Computer Supported Collaborative Working**: An Epi-Collaborator can be thought of as a recommender system, thereby drawing in additional expertise from an area of much recent interest and activity [2]. My team and I have recently used recommender-based approaches for pricing crowdsourced software engineering tasks and identifying suitable developers to undertake them, based on their skills [94, 95]. Recommender system research represents another opportunity for fruitful cross-fertilisation of ideas, methods and techniques, between the EPIC project and other related research communities.

I plan to extend my work with Licia Capra, another full professor in my group, with whom I have recently been collaborating on crowdsourcing [93], benefiting from her expertise in Recommender Systems [88] and Computer Supported Cooperative Work (CSCW) [39, 97, 116]. Through EPIC, I will work with UCL-based colleagues, and the wider community to draw on recent results in recommender-based systems and CSCW, investigating novel algorithms, approaches and techniques developed by that community.

As with the relationships to other areas of software engineering, the relationship to recommender systems will be a two-way partnership, in which the EPIC project will both draw on and contribute to developments in the field. In particular, EPIC's approaches to automatically executing systematic experiments to explain, justify and document recommendations will develop techniques that will cross over into other domains of application for recommender-based systems.

**Relationships with Software Testing**: Automated software test data generation will be an important aspect of the EPIC project throughout all activities, and will have a role to play in many of the techniques we shall develop. We need high quality test cases to ensure faithfulness to the semantics to be preserved, avoiding regression, and also to investigate the performance of the improvements against the non-functional properties of interest.

EPIC would also use test suites to document and justify the suggestions made by the Epi-Collaborators, and will use the existing system to be improved as a source of valuable oracle information. Fortunately, there have been dramatic advances in automated software test input generation, on which the project will be able to build [27, 65].

Epi-Collaborators offer new opportunities to develop advanced testing techniques, such as competitive co-evolutionary testing, which I proposed for work package WP4. The transplants and new features suggested by collaborators will also be used as the basis for new forms of software test data generation. Specifically, EPIC will investigate a transplant-based test case generation approach. For a given feature, $F$, to be tested in a host system, we will use Epi-Collaborators to identify multiple 'near-miss' donor transplants for $F$. We will use counterexample-driven test data generation, to generate test cases that distinguish between these near-miss donor transplants and the original program.

This will introduce a new form of higher order mutation testing [63, 64], in which the near-miss donor transplant plays the role of a higher order mutant, constructed to be a semantic near-miss to the feature as implemented by the developer. The research hypothesis we will investigate in this avenue of the project, is that test cases that can distinguish between such a near-miss feature and the true feature will also be good at uncovering faults. The key insight is that a near-miss feature captures the kind of 'subtle near-miss behaviour' between a faulty program and the correct version [110].

**Relationships with Evolutionary Computation**: Evolutionary computation lies at the heart of the EPIC project. Naturally, I expect to draw on developments and advances in this field. However, I also believe that the EPIC approach can help to reinvigorate research in evolutionary computing in general, and in genetic programming in particular. Our recent breakthroughs in genetic improvement [18, 86, 113] have demonstrated this possibility, achieving human competitive results on real-world software systems using variations and augmentations of genetic programming.

**Relationships with Software Agents**: Although Search Based Software Engineering has been applied to autonomous software agents [77, 106], an Epi-Collaborator is not an *autonomous* agent, in the sense usually intended by the Software Agents community [52, 75], because it deliberately leaves all critical decision-making to the human engineer (thereby lacking full autonomy). Nevertheless, there are obvious intellectual connections between the EPIC project vision and agent-based ideas, so the project will naturally seek to learn from developments in Software Agents, incorporating them where appropriate.

# 1 Note: written in November 2018

I am often asked about funding proposals and believe in sharing them, where possible, so here is this one (minus confidential details). Please be aware that it was written in 2016 and ideas change over time.

I've had the good fortune to spend some time as a full timer at Facebook since the start of 2017, working on practical, scalable software engineering. That experience has, naturally, changed some of my ideas and plans. Nevertheless, I remain completely convinced by the central theme and approach set out in this proposal. I plan to dedicate myself and my scientific work to it.

If anything, my experience of scale and real world challenges in the tech sector has underscored the need for this approach to build the next generation of intelligent, empirically experimenting, software engineering bots. This is a direction of fundamental importance for automated software engineering in my view.

That is, we need to develop of ways to build bots that talk to us human developers in actionable ways that export bots' considerable, *but highly peculiar*, 'skill set', and to build the interfaces that complement human skill sets, so that Software Engineering can benefit from the collection of techniques and opportunities that combine optimisation, computational search, and Machine learning.

I believe that this will enable us to best draw on the resources of data and compute power with which such a bot can become a really efficient, effective, dedicated empirical software engineering experimentor. the result of these experiments will be scientifically-justified suggestions for improving softer systems.

## References


[1] K. Adamopoulos, M. Harman, and R. M. Hierons. Mutation testing using genetic algorithms: A co-evolution approach. In *Genetic and Evolutionary Computation Conference (GECCO 2004), LNCS 3103*, pages 1338–1349, Seattle, Washington, USA, June 2004. Springer.

[2] Adomavicius and Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17, 2005.

[3] W. Afzal and R. Torkar. On the application of genetic programming for software engineering predictive modeling: A systematic review. *Expert Systems Applications*, 38(9):11984–11997, 2011.

[4] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[5] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test-case generation. *IEEE Transactions on Software Engineering*, pages 742–762, 2010.

[6] N. Alshahwan and M. Harman. Automated web application testing using search based software engineering. In $26^{th}$ *IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 3 – 12, Lawrence, Kansas, USA, 6th - 10th November 2011.

[7] S. Anand, A. Bertolino, E. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, J. Li, P. McMinn, and H. Zhu. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, August 2013.

[8] K. Androutsopoulos, D. Binkley, D. Clark, N. Gold, M. Harman, K. Lano, and Z. Li. Model projection: Simplifying models in response to restricting the environment. In $33^{rd}$ *International Conference on Software Engineering (ICSE '11)*, pages 291–300, New York, NY, USA, 2011. ACM.

[9] K. Androutsopoulos, D. Clark, H. Dan, M. Harman, and R. Hierons. An analysis of the relationship between conditional entropy and failed error propagation in software testing. In $36^{th}$ *International Conference on Software Engineering (ICSE 2014)*, pages 573–583, Hyderabad, India, June 2014.

[10] K. Androutsopoulos, D. Clark, M. Harman, J. Krinke, and L. Tratt. State-based model slicing: A survey. *ACM Computing Surveys*, 45(4):53:1–53:36, Aug. 2013.

[11] A. Arcuri, D. R. White, J. A. Clark, and X. Yao. Multi-objective improvement of software using co-evolution and smart seeding. In X. Li, M. Kirley, M. Zhang, D. G. Green, V. Ciesielski, H. A. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, and Y. Shi, editors, $7^{th}$ *International Conference on Simulated Evolution and Learning (SEAL 2008)*, volume 5361 of *Lecture Notes in Computer Science*, pages 61–70, Melbourne, Australia, December 2008. Springer.

[12] A. Arcuri and X. Yao. A Novel Co-evolutionary Approach to Automatic Software Bug Fixing. In *IEEE Congress on Evolutionary Computation (CEC'08)*, pages 162–168, Hongkong, China, 1-6 June 2008. IEEE Computer Society.

[13] A. Baars, M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, P. Tonella, and T. Vos. Symbolic search-based testing. In $26^{th}$ *IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 53 – 62, Lawrence, Kansas, USA, 6th - 10th November 2011.

[14] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In $22^{nd}$ *International Conference on Software Maintenance (ICSM 06)*, pages 176–185, Philadelphia, Pennsylvania, USA, Sept. 2006.

[15] F. Balmas. Using dependence graphs as a support to document programs. In $2^{nd}$ *International Workshop on Source Code Analysis and Manipulation*, pages 145–154, Montreal, Canada, 2002. IEEE Computer Society.

[16] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, Jan. 1998.

[17] E. T. Barr, Y. Brun, P. Devanbu, M. Harman, and F. Sarro. The plastic surgery hypothesis. In $22^{nd}$ *ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, pages 306–317, Hong Kong, China, November 2014.

[18] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke. Automated software transplantation (gold medal winner at GECCO 2016 human competitive results competition — The HUMIES. Also winner of an ACM distinguished paper award at ISSTA 2015). In *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*, pages 257–269, 2015.

[19] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, May 2015.

[20] I. D. Baxter. Transformation systems: Domain-oriented component and implementation knowledge. In $9^{th}$ *Workshop on Institutionalizing Software Reuse*, Austin, TX, USA, Jan. 1999.

[21] A. Bertolino, G. D. Angelis, L. Frantzen, and A. Polini. Model-based generation of testbeds for web services. In K. Suzuki, T. Higashino, A. Ulrich, and T. Hasegawa, editors, $8^{th}$ *International*

*Workshop on Formal Approaches to Testing Software (FATES '08)*, volume 5047 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2008.

[22] D. Binkley, N. Gold, M. Harman, S. Islam, J. Krinke, and S. Yoo. ORBS: Language-independent program slicing. In $22^{nd}$ *ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*, pages 109–120, Hong Kong, China, November 2014.

[23] D. Binkley and M. Harman. Analysis and visualization of predicate dependence on formal parameters and global variables. *IEEE Transactions on Software Engineering*, 30(11):715–735, 2004.

[24] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In W. Tracz, M. P. Robillard, and T. Bultan, editors, *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*. ACM, 2012. Article 45(1-10).

[25] S. Bouktif, G. Antoniol, E. Merlo, and M. Neteler. A novel approach to optimize clone refactoring activity. In M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, and D. Thierens, editors, $8^{th}$ *annual conference on Genetic and evolutionary computation (GECCO 2006)*, volume 2, pages 1885–1892, Seattle, Washington, USA, 8-12 July 2006. ACM Press.

[26] B. Bruce, J. Petke, and M. Harman. Reducing energy consumption using genetic improvement. In *Genetic and evolutionary computation conference (GECCO 2015)*, pages 1327–1334, Madrid, Spain, July 2015.

[27] C. Cadar, P. Godefroid, S. Khurshid, C. S. Păsăreanu, K. Sen, N. Tillmann, and W. Visser. Symbolic execution for software testing in practice: preliminary assessment. In $33^{rd}$ *International Conference on Software Engineering (ICSE'11)*, pages 1066–1071, New York, NY, USA, 2011. ACM.

[28] C. Cadar, P. Pietzuch, and A. L. Wolf. Multiplicity computing: a vision of software engineering for next-generation computing platform applications. In *Workshop on Future of Software Engineering Research (FoSER 2010)*, pages 81–86. ACM, 2010.

[29] C. Cadar, P. Pietzuch, and A. L. Wolf. Multiplicity computing: a vision of software engineering for next-generation computing platform applications. In G.-C. Roman and K. J. Sullivan, editors, *Workshop on Future of Software Engineering Research (FoSER 2010)*, pages 81–86. ACM, 2010.

[30] B. Cheng and J. Atlee. From state of the art to the future of requirements engineering. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, Los Alamitos, California, USA, 2007. IEEE Computer Society Press. This volume.

[31] J. F. Chicano, J. Ferrer, and E. Alba. Elementary landscape decomposition of the test suite minimization problem. In M. B. Cohen and M. Ó Cinnédie, editors, $3^{rd}$ *International Symposium on Search Based Software Engineering (SSBSE 2011)*, volume 6956 of *Lecture Notes in Computer Science*, pages 48–63, Szeged, Hungary, 2011. Springer.

[32] D. Clark, C. Hankin, and S. Hunt. Information flow for algol-like languages. *Computer Languages*, 28(1):3–28, 2002.

[33] J. Cleland-Huang, C. K. Chang, and M. J. Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[34] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2001.

[35] J. R. Cordy. The TXL source transformation language. *Science of Computer Programming*, 61(3):190–210, 2006.

[36] S. L. Cornford, M. S. Feather, J. R. Dunphy, J. Salcedo, and T. Menzies. Optimizing Spacecraft Design - Optimization Engine Development: Progress and Plans. In *Proceedings of the IEEE Aerospace Conference*, pages 3681–3690, Big Sky, Montana, March 2003.

[37] S. Danicic, R. W. Barraclough, M. Harman, J. Howroyd, Á. Kiss, and M. R. Laurence. A unifying theory of control dependence and its application to arbitrary program structures. *Theoretical Computer Science*, 412(49):6809–6842, 2011.

[38] A. De Lucia. Program slicing: Methods and applications. In $1^{st}$ *IEEE International Workshop on Source Code Analysis and Manipulation*, pages 142–149, Los Alamitos, California, USA, 2001. IEEE Computer Society Press.

[39] M. Dittus, G. Quattrone, and L. Capra. Analysing volunteer engagement in humanitarian mapping: Building contributor communities at large scale. In D. Gergle, M. R. Morris, P. Bjørn, and J. A. Konstan, editors, *19th ACM Conference on Computer-Supported Cooperative Work & Social Computing (CSCW 2016)*, pages 108–118, San Francisco, CA, USA, 2016. ACM.

[40] V. D'Silva, D. Kroening, and G. Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(7):1165–1178, 2008.

[41] E. Engström, P. Runeson, and M. Skoglund. A systematic review on regression test selection techniques. *Information & Software Technology*, 52(1):14–30, 2010.

[42] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In $21^{st}$ *International Conference on Software Engineering (ICSE-99)*, pages 213–225, NY, May 16–22 1999. ACM Press.

[43] D. Fatiregun, M. Harman, and R. Hierons. Search based transformations. In E. Cant-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation – GECCO-2003*, volume 2724 of *LNCS*, pages 2511–2512, Berlin, 12-16 July 2003. Springer-Verlag.

[44] F. Ferrucci, M. Harman, and F. Sarro. Search based software project management. In G. Ruhe and C. Wohlin, editors, *Software Project Management in a Changing World*. Springer, 2014. To appear.

[45] A. Finkelstein, M. Harman, A. Mansouri, J. Ren, and Y. Zhang. Fairness analysis in requirements assignments. In $16^{th}$ *IEEE International Requirements Engineering Conference*, pages 115–124, Los Alamitos, California, USA, September 2008. IEEE Computer Society Press.

[46] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, Los Alamitos, California, USA, 2007. IEEE Computer Society Press.

[47] G. Fraser and A. Arcuri. EvoSuite: automatic test suite generation for object-oriented software. In $8^{th}$ *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11)*, pages 416–419. ACM, September 5th - 9th 2011.

[48] F. G. Freitas and J. T. Souza. Ten years of search based software engineering: A bibliometric analysis. In $3^{rd}$ *International Symposium on Search based Software Engineering (SSBSE 2011)*, pages 18–32, 10th - 12th September 2011.

[49] Z. P. Fry, B. Landau, and W. Weimer. A human study of patch maintainability. In *International Symposium on Software Testing and Analysis (ISSTA'12)*, pages 177–187, Minneapolis, Minnesota, USA, July 2012.

[50] M. Gabel and Z. Su. A study of the uniqueness of source code. In $18^{th}$ *ACM SIGSOFT international symposium on foundations of software engineering (FSE 2010)*, pages 147–156, Santa Fe, New Mexico, USA, 7-11 Nov. 2010. ACM.

[51] V. Garousi, L. C. Briand, and Y. Labiche. Traffic-aware stress testing of distributed systems based on UML models. In *International Conference on Software Engineering (ICSE '06)*, pages 391–400, 2006.

[52] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[53] R. Giacobazzi and I. Mastroeni. Non–standard semantics for program slicing. *Higher-Order and Symbolic Computation*, 16(4):297–339, 2003. Special issue on Partial Evalution and Semantics-Based Program Manipulation.

[54] B. Godlin and O. Strichman. Regression verification: proving the equivalence of similar programs. *Software Testing, Verification and Reliability*, 23(3):241–258, 2013.

[55] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[56] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *First International Conference on Requirements Engineering (ICRE'94)*, pages 94–101, Apr. 1994.

[57] D. Greer and G. Ruhe. Software release planning: an evolutionary and iterative approach. *Information & Software Technology*, 46(4):243–253, 2004.

[58] M. Harman. The current state and future of search based software engineering. In L. Briand and A. Wolf, editors, *Future of Software Engineering 2007*, pages 342–357, Los Alamitos, California, USA, 2007. IEEE Computer Society Press.

[59] M. Harman. Software engineering meets evolutionary computation. *IEEE Computer*, 44(10):31–39, Oct. 2011.

[60] M. Harman. The role of artificial intelligence in software engineering (keynote paper). In $1^{st}$ *International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE 2012)*, Zurich, Switzerland, 2012.

[61] M. Harman, L. Hu, R. M. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper. Testability transformation. *IEEE Transactions on Software Engineering*, 30(1):3–16, Jan. 2004.

[62] M. Harman, Y. Jia, J. Krinke, B. Langdon, J. Petke, and Y. Zhang. Search based software engineering for software product line engineering: a survey and directions for future work (keynote paper). In $18^{th}$ *International Software Product Line Conference (SPLC 14)*, pages 5–18, Florence, Italy, September 2014.

[63] M. Harman, Y. Jia, and W. B. Langdon. Strong higher order mutation-based test data generation. In $8^{th}$ *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11)*, pages 212–222, New York, NY, USA, September 5th - 9th 2011. ACM.

[64] M. Harman, Y. Jia, P. R. Mateo, and M. Polo. Angels and monsters: an empirical investigation of potential test effectiveness and efficiency improvement from strongly subsuming higher order mutation. In *ACM/IEEE International Conference on Automated Software Engineering (ASE '14)*, pages 397–408, 2014.

[65] M. Harman, Y. Jia, and Y. Zhang. Achievements, open problems and challenges for search based software testing (keynote paper). In $8^{th}$ *IEEE International Conference on Software Testing, Verification and Validation (ICST 2015)*, Graz, Austria, April 2015.

[66] M. Harman and B. F. Jones. Search based software engineering. *Information and Software Technology*, 43(14):833–839, Dec. 2001.

[67] M. Harman, W. B. Langdon, and Y. Jia. Babel pidgin: SBSE can grow and graft entirely new functionality into a real world system (winner of SBSE Challenge 2014 at SSBSE 2014. In $6^{th}$ *Symposium on Search Based Software Engineering (SSBSE 2014)*, pages 247–252, Fortaleza, Brazil, August 2014. Springer LNCS.

[68] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark. The GISMOE challenge: Constructing the pareto program surface using genetic programming to find better programs (keynote paper). In $27^{th}$ *IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*, pages 1–14, Essen, Germany, September 2012.

[69] M. Harman, A. Mansouri, and Y. Zhang. Search based software engineering: Trends, techniques and applications. *ACM Computing Surveys*, 45(1):11:1–11:61, November 2012.

[70] M. Harman and P. McMinn. A theoretical and empirical study of search based testing: Local, global and hybrid search. *IEEE Transactions on Software Engineering*, 36(2):226–247, 2010.

[71] W. Heaven and E. Letier. Simulating and optimising design decisions in quantitative goal models. In $19^{th}$ *IEEE International Requirements Engineering Conference (RE 2011)*, pages 79–88. IEEE, 2011.

[72] C. Henard, M. Papadakis, M. Harman, and Y. L. Traon. Combining multi-objective search and constraint solving for configuring large software product lines. In $37^{th}$ *International Conference on Software Engineering (ICSE 2015)*, pages 517–528, Florence, Italy, 2015.

[73] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 25–46, Atlanta, Georgia, June 1988. Proceedings in *SIGPLAN Notices*, 23(7), pp.35–46, 1988.

[74] K. Inkumsah and T. Xie. Evacon: a framework for integrating evolutionary and concolic testing for object-oriented programs. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, pages 425–428. ACM, 2007.

[75] N. R. Jennings and M. J. Wooldridge. Software agents. *IEE review*, pages 17–20, 1996.

[76] Y. Jia and M. Harman. Milu: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In $3^{rd}$ *Testing Academia and Industry Conference - Practice and Research Techniques (TAIC PART'08)*, pages 94–98, Windsor, UK, August 2008.

[77] S. Kalboussi, S. Bechikh, M. Kessentini, and L. B. Said. On the influence of the number of objectives in evolutionary autonomous software agent testing. In *25th International Conference on Tools with Artificial Intelligence (ICTAI '13)*, pages 229–234, Herndon, VA, USA, 4-6 November 2013. IEEE.

[78] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, September/October 1997.

[79] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[80] J. Krinke and G. Snelting. Validation of measurement software as an application of slicing and constraint solving. *Information and Software Technology Special Issue on Program Slicing*, 40(11 and 12):661–675, 1998.

[81] K. Lakhotia, M. Harman, and H. Gross. AUSTIN: An open source tool for search based software testing of C programs. *Journal of Information and Software Technology*, 55(1):112–125, January 2013.

[82] K. Lakhotia, N. Tillmann, M. Harman, and J. de Halleux. FloPSy — Search-based floating point constraint solving for symbolic execution. In $22^{nd}$ *IFIP International Conference on Testing Software and Systems (ICTSS 2010)*, pages 142–157, Natal, Brazil, November 2010. LNCS Volume 6435.

[83] W. B. Langdon and M. Harman. Evolving a CUDA kernel from an nVidia template. In *IEEE Congress on Evolutionary Computation*, pages 2376–2383. IEEE, 2010.

[84] W. B. Langdon and M. Harman. Genetically improved CUDA C++ software. In $17^{th}$ *European Conference on Genetic Programming (EuroGP)*, pages 84–95, Granada, Spain, April 2014.

[85] W. B. Langdon and M. Harman. Grow and graft a better CUDA pknotsrg for RNA pseudoknot free energy calculation. In *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*, pages 805–810, 2015.

[86] W. B. Langdon and M. Harman. Optimising existing software with genetic programming. *IEEE Transactions on Evolutionary Computation (TEVC)*, 19(1):118–135, Feb 2015.

[87] W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer, 2002.

[88] N. Lathia, S. Hailes, L. Capra, and X. Amatriain. Temporal diversity in recommender systems. In F. Crestani, S. Marchand-Maillet, H.-H. Chen,

E. N. Efthimiadis, and J. Savoy, editors, *33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2010)*, pages 210–217, Geneva, Switzerland, 2010. ACM.

[89] C. Le Goues, S. Forrest, and W. Weimer. Current challenges in automatic software repair. *Software Quality Journal*, 21(3):421–443, 2013.

[90] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. GenProg: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012.

[91] D. Li, A. H. Tran, and W. G. J. Halfond. Making web applications more energy efficient for OLED smartphones. In *36th International Conference on Software Engineering (ICSE 2014)*, pages 527–538, New York, NY, USA, 2014. ACM.

[92] I. Manotas, L. Pollock, and J. Clause. SEEDS: A software engineer's energy-optimization decision support framework. In *36th International Conference on Software Engineering*, pages 503–514, New York, NY, USA, 2014. ACM.

[93] K. Mao, L. Capra, M. Harman, and Y. Jia. A survey of the use of crowdsourcing in software engineering. Technical Report RN/15/01, UCL Computer Science Department, May 2015.

[94] K. Mao, Y. Yang, M. Li, and M. Harman. Pricing crowdsourcing-based software development tasks. In $35^{th}$ *ACM/IEEE International Conference on Software Engineering (ICSE 2013 — NIER track)*, San Francisco, USA, 2013.

[95] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman. Developer recommendation for crowdsourced software development tasks. In *2015 IEEE Symposium on Service-Oriented System Engineering, SOSE 2015, San Francisco Bay, CA, USA, March 30 - April 3, 2015*, pages 347–356, 2015.

[96] W. Martin, F. Sarro, and M. Harman. Causal impact analysis for app releases in Google Play. In *24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering(FSE 2016)*, pages 435–446, Seattle, WA, USA, November 2016.

[97] A. J. Mashhadi, G. Quattrone, and L. Capra. Putting ubiquitous crowd-sourcing into context. In A. Bruckman, S. Counts, C. Lampe, and L. G. Terveen, editors, *Computer Supported Cooperative Work (CSCW 2013)*, pages 611–622, San Antonio, TX, USA, 2013. ACM.

[98] S. Mcilroy, N. Ali, and A. E. Hassan. Fresh apps: An empirical study of frequently-updated mobile apps in the google play store. *Empirical Software Engineering*, 21(3):1346–1370, June 2016.

[99] P. McMinn. Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2):105–156, June 2004.

[100] J. F. Miller, editor. *Cartesian Genetic Programming*. Natural Computing Series. Springer, 2011.

[101] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[102] B. S. Mitchell and S. Mancoridis. On the automatic modularization of software systems using the bunch tool. *IEEE Transactions on Software Engineering*, 32(3):193–208, 2006.

[103] I. H. Moghadam and Mel Ó Cinnéide. Code-Imp: A tool for automated search-based refactoring. In *Proceeding of the 4th workshop on Refactoring Tools (WRT '11)*, pages 41–44, Honolulu, HI, USA, 2011.

[104] A. Moraglio, K. Krawiec, and C. G. Johnson. Geometric semantic genetic programming. In C. A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, editors, *Parallel Problem Solving from Nature, PPSN XII (part 1)*, volume 7491 of *Lecture Notes in Computer Science*, pages 21–31, Taormina, Italy, Sept. 1-5 2012. Springer.

[105] A. Ngo-The and G. Ruhe. A systematic approach for solving the wicked problem of software release planning. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 12(1):95–108, August 2008.

[106] C. Nguyen, A. Perini, P. Tonella, S. Miles, M. Harman, and M. Luck. Evolutionary testing of autonomous software agents. In $8^{th}$ *International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 521–528, Budapest, Hungary, May 2009.

[107] H. D. T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra. SemFix: program repair via semantic analysis. In B. H. C. Cheng and K. Pohl, editors, *35th International Conference on Software Engineering (ICSE 2013)*, pages 772–781, San Francisco, USA, May 18-26 2013. IEEE.

[108] P. Nordin. *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, der Universitat Dortmund am Fachereich Informatik, 1997.

[109] M. Ó Cinnédie, L. Tratt, M. Harman, S. Counsell, and I. H. Moghadam. Experimental assessment of software metrics using automated refactoring (best paper award winner). In $6^{th}$ *IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2012)*, pages 49–58, Lund, Sweden, September 2012.

[110] E. Omar, S. Ghosh, and D. Whitley. Constructing subtle higher order mutants for Javer and AspectJ programs. In *International Symposium on Software Reliability Engineering (ISSRE'13)*, pages 340–349. IEEE, 2013.

[111] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, Aug. 2001.

[112] M. Orlov and M. Sipper. Flight of the FINCH through the java wilderness. *IEEE Transactions Evolutionary Computation*, 15(2):166–182, 2011.

[113] J. Petke, M. Harman, W. B. Langdon, and W. Weimer. Using genetic improvement & code transplants to specialise a C++ program to a problem class (silver medal winner at GECCO 2014 human competitive results competition — The HUMIES. In $17^{th}$ *European Conference on Genetic Programming (EuroGP)*, pages 132–143, Granada, Spain, April 2014.

[114] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[115] Z. Qi, F. Long, S. Achour, and M. Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *International Symposium on Software Testing and Analysis (ISSTA 2015)*, pages 24–36, New York, NY, USA, 2015. ACM.

[116] G. Quattrone, L. Capra, and P. D. Meo. There's no such thing as the perfect map: Quantifying bias in spatial crowd-sourcing datasets. In D. Cosley, A. Forte, L. Ciolfi, and D. McDonald, editors, *18th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW 2015)*, pages 1021–1032, Vancouver, BC, Canada, 2015. ACM.

[117] O. Räihä. A survey on search–based software design. *Computer Science Review*, 4(4):203–249, 2010.

[118] V. P. Ranganath, T. Amtoft, A. Banerjee, J. Hatcliff, and M. B. Dwyer. A new foundation for control dependence and slicing for modern program structures. *ACM Transactions on Programming Languages and Systems*, 29(5), 2007.

[119] J. Ren, M. Harman, and M. Di Penta. Cooperative co-evolutionary optimization on software project staff assignments and job scheduling. In $3^{rd}$ *International Symposium on Search based Software Engineering (SSBSE 2011)*, pages 127–141, 10th - 12th September 2011. LNCS Volume 6956.

[120] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, 27(10):929–948, Oct. 2001.

[121] C. Ryan. *Automatic re-engineering of software using genetic programming*. Kluwer Academic Publishers, 2000.

[122] R. A. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J. Harrold. Test-suite augmentation for evolving software. In $23^{rd}$ *Automated Software Engineering (ASE '08)*, pages 218–227, L'Aquila, Italy, 2008. IEEE.

[123] F. Sarro, A. A. Al-Subaihin, M. Harman, Y. Jia, W. Martin, and Y. Zhang. Feature lifecycles as they spread, migrate, remain, and die in app stores. In *23rd IEEE International Requirements Engineering Conference, RE 2015, Ottawa, ON, Canada, August 24-28, 2015*, pages 76–85, 2015.

[124] E. Schulte, J. Dorn, S. Harding, S. Forrest, and W. Weimer. Post-compiler software optimization for reducing energy. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 639–652, Salt Lake City, UT, USA, 2014. ACM.

[125] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. C. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In T. Gyimóthy and A. Zeller, editors, $19^{th}$ *ACM Symposium on the Foundations of Software Engineering (FSE-19)*, pages 124–134, Szeged, Hungary, Sept. 2011. ACM.

[126] P. Sitthi-amorn, N. Modly, W. Weimer, and J. Lawrence. Genetic programming for shader simplification. *ACM Transactions on Graphics*, 30(6):152:1–152:11, 2011.

[127] P. Somavat and V. Namboodiri. Energy consumption of personal computing including portable communication devices. *Journal of Green Engineering*, 1(4):447–475, 2011.

[128] M. Staats, G. Gay, and M. P. E. Heimdahl. Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing. In $34^{th}$ *International Conference on Software Engineering (ICSE 2012)*, pages 870–880, 2012.

[129] L. H. Tahat, B. Korel, M. Harman, and H. Ural. Regression test suite prioritization using system models. *Journal of Software Testing, Verification and Reliability*, 22(7):481–506, 2012.

[130] N. Tillmann, J. de Halleux, and T. Xie. Transferring an automated test generation tool to practice: From Pex to Fakes and Code Digger. In *29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*, pages 385–396, 2014.

[131] P. Tonella. Evolutionary testing of classes. In *International Symposium on Software Testing and Analysis (ISSTA '04)*, pages 119–128, Boston, Massachusetts, USA, 11-14 July 2004. ACM.

[132] P. Tonella, A. Marchetto, D. C. Nguyen, Y. Jia, K. Lakhotia, and M. Harman. Finding the optimal balance between over and under approximation of models inferred from execution logs. In $5^{th}$ *International Conference on Software Testing, Verification and Validation (ICST 2012)*, pages 21–30, Montreal, Canada, 2012.

[133] A. van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[134] E. Visser. A survey of strategies in rule-based program transformation systems. *Journal of Symbolic Computation*, 40(1):831–873, 2005.

[135] J. A. Walker and J. F. Miller. The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Transactions on*

*Evolutionary Computation*, 12(4):397–417, Aug. 2008.

[136] B. Walsh. The surprisingly large energy footprint of the digital economy. *Time Magazine*, August 14th 2013.

[137] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. *Information and Software Technology*, 43(14):841–854, 2001.

[138] D. R. White, A. Arcuri, and J. A. Clark. Evolutionary improvement of programs. *IEEE Transactions on Evolutionary Computation (TEVC)*, 15(4):515–538, 2011.

[139] D. R. White, J. Clark, J. Jacob, and S. Poulding. Searching for resource-efficient programs: Low-power pseudorandom number generators. In *2008 Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 1775–1782, Atlanta, USA, July 2008.

[140] D. Whitley, A. M. Sutton, and A. E. Howe. Understanding elementary landscapes. In $10^{th}$ *annual conference on Genetic and evolutionary computation (GECCO 2008)*, pages 585–592, New York, NY, USA, 2008. ACM.

[141] F. Wu, M. Harman, Y. Jia, J. Krinke, and W. Weimer. Deep parameter optimisation. In *Genetic and evolutionary computation conference (GECCO 2015)*, pages 1375–1382, Madrid, Spain, July 2015.

[142] S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. *Journal of Software Testing, Verification and Reliability*, 22(2):67–120, 2012.

[143] S. Yoo and M. Harman. Test data regeneration: Generating new test data from existing test data. *Journal of Software Testing, Verification and Reliability*, 22(3):171–201, May 2012.

[144] S. Yoo, M. Harman, and D. Clark. Fault localization prioritization: Comparing information theoretic and coverage based approaches. *ACM Transactions on Software Engineering and Methodology*, 22(3 (Article 19)), July 2013.

[145] S. Yoo, R. Nilsson, and M. Harman. Faster fault finding at Google using multi objective regression test optimisation. In $8^{th}$ *European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE '11)*, Szeged, Hungary, September 5th - 9th 2011. Industry Track.

[146] Y. Zhang, E. Alba, J. J. Durillo, S. Eldh, and M. Harman. Today/future importance analysis. In *ACM Genetic and Evolutionary Computation COnference (GECCO 2010)*, pages 1357–1364, Portland Oregon, USA, 7th–11th July 2010.

[147] Y. Zhang, M. Harman, Y. Jia, and F. Sarro. Inferring test models from Kate's bug reports using multi-objective search (SBSE Challenge 2016 Winner). In $7^{th}$ *International Symposium on Search-Based Software Engineering (SSBSE)—Challenge Track*, pages 301–307, Bergamo, Italy, September 2015.