Search based software engineering for software product line engineering: a survey and directions for future work

M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke & Y. Zhang
CREST Centre, University College London, Malet Place, London, WC1E 6BT, U.K.

ABSTRACT

This paper[1] presents a survey of work on Search Based Software Engineering (SBSE) for Software Product Lines (SPLs). We have attempted to be comprehensive, in the sense that we have sought to include all papers that apply computational search techniques to problems in software product line engineering. Having surveyed the recent explosion in SBSE for SPL research activity, we highlight some directions for future work. We focus on suggestions for the development of recent advances in genetic improvement, showing how these might be exploited by SPL researchers and practitioners: Genetic improvement may grow new products with new functional and non-functional features and graft these into SPLs. It may also merge and parameterise multiple branches to cope with SPL branchmania.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.11 [**Software Engineering**]: Software Architectures

Keywords

SBSE, SPL, Genetic Programming, Program Synthesis

1. INTRODUCTION

A Software Product Line (SPL) [113, 124], is a collection of related software products, all of which share some core functionality, yet each of which differs in some specific features. These differences in the features offered by each product help the product line to capture the variability required by different users, different platforms and different operating environments. The purpose of SPL engineering is to optimally manage the elicitation/extraction, analysis, evolution and application of the feature model and the SPL architecture and the products constructed from them.

[1]The paper was written to accompany the keynote at the 15^{th} Software Product Line Conference (SPLC 2014), given by Mark Harman, but it reflects the joint work of all the authors.

A well managed SPL benefits from the commonalities shared by all features. It allows us to address many of the goals that have long been sought in software engineering, such as reuse, traceability of requirements into products, systematic testing and the control of software maintenance and evolutionary processes.

It is natural to think of many of the problems in software product line engineering as problems of *optimisation*. The wide variability of different products expressed by their feature models, creates a large and rich search space in which we can seek optimal (or near optimal) choices of products. Many of the problems in SPL engineering involve balances of multiple competing and conflicting software engineering concerns.

Problems such as this, characterised by a large search space, in which we seek to satisfy multiple conflicting or competing objectives and constraints, are not unique to SPL engineering alone. Such software engineering problems have repeatedly been shown to submit to search-based formulation and solution using techniques from the area that has come to be known as 'Search Based Software Engineering' (SBSE) [48, 65].

The term 'Search Based Software Engineering', coined in 2001 [60], refers to the use of computational search as a means of optimising software engineering problems. Since 2001, SBSE has been used to address challenging software engineering problems with large and complex search spaces, characterised by many conflicting competing objectives, in areas as diverse as requirements [162], predictive modelling [1, 55] software project management [43], design [134], testing [2, 112], refactoring [125] and repair [93]. This wide applicability to problems characterised by similar features to those found in SPL engineering, naturally suggests that SBSE will find successful applications in SPL optimisation. This observation has led to a recent upsurge in interest and activity in the area of 'SBSE for SPL'.

Figure 1 shows the growth in papers on SBSE for SPL. The number of papers expected for 2014 is estimated, based on those we found at the time of writing (end of May 2014). The 2014 figure is estimated by simply multiplying those papers seen in the first five months of the year by 12/5 to linearly extend. While we include some 'to appear' papers for 2014, there will also be an inherent lag in paper indexing. Bearing in mind these two conflicting confounders, we believe a simple linear extrapolation is the simplest approach. Naturally, this number should be treated with caution. Whatever the true result for 2014, the growth trend reveals a large upsurge in interest in the past five years.

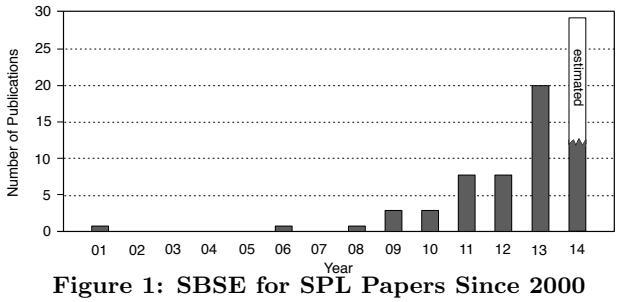

Figure 1: SBSE for SPL Papers Since 2000

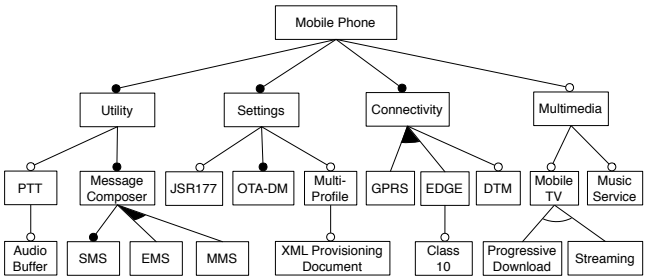

Figure 2: Extract from a 2005 Motorola Phone SPL

As well as the many subject specific SBSE surveys, there are several publicly available SBSE tools for release planning [121], design [115], testing [4, 45, 51, 78, 88, 147], refactoring [117] and repair [94]. However, hitherto, there has been no survey of SBSE for SPL. This next section sets out to provide a comprehensive survey of this literature.

2. SBSE OF SPL ENGINEERING SURVEY

This section provides a survey of work on search based software engineering for software product lines. At the heart of most SPLs lies the feature model. Much of the work on SBSE for SPLs is concerned with the problem of extracting and improving the feature model, and selecting and prioritising the choice of products to be constructed from it.

The first two sections (Sections 2.1 and 2.2) concern the problems of optimising feature model construction and product selection/prioritisation from these models. A slightly less well studied, but nonetheless very important and promising area, concerns improvement and repair of software product line architectures. This is covered in the third section (Section 2.3). Finally, there has been a great deal of work on testing software product lines. The final section (Section 2.4) covers test case construction, selection and prioritisation, which has been attacked using many different search based optimisation approaches, but most notably search based combinatorial interaction testing.

2.1 Search Based Feature Model Selection

The concept of a feature model dates back to the work of Kang et al. [82], who first formulated the notion of domain features as an important space to be modelled, analysed and controlled. There are many languages and notations available for describing feature models (also known as variability models), such as TVL [27], DOPLER [38], FODA [19], and commercial tools and tool extensions such as Gears [12]. A survey of feature model diagrammatic notations can be found elsewhere [141].

These notations represent the common features, their characteristics, defaults, special features and constraints. From such models, developers can both derive specific instantiations (products), and also manage, analyse and understand the interactions and constraints pertinent to their product line features.

As an illustrative example of a feature model diagram, consider the model depicted in Figure 2. The model is taken from a real 2005 Motorola mobile phone product line, widely used as a dataset in search based requirements optimisation since 2006 [13]. The fact that a feature model can simultaneously be used to study feature model optimisation *and* requirements optimisation, underscores the very close relationship between two problems.

In this feature model, SMS is the Short Message Service (aka text messaging), which remains ubiquitous ten years later. EMS is an SMS concatenation protocol, while MMS is the Multimedia Message Service protocol that includes pictures (and, on more recent phones, video). PTT is a Push To Talk feature that provides an instant communication solution, with optional audio buffering facility. Multi-Profile allows several users to share the same phone, and includes an XML export format. OTA-DM is Over The Air Device Management, which allows the provider remove device configuration access. JSR177 is Java Specification Request 177, which provides security and trust functionality for J2ME. GPRS is the General Packet Radio Services low bandwidth data connection, while EDGE provides enhanced bandwidth, for which 'Class 10' is a specific optional version. DTM is Dual Transfer Mode communication between device and base station.

Real world feature models typically involve many constraints [17]. Tracing variability information between problems (requirements) and solutions (products) is challenging [16]. Languages, notations and tools (such as TVL, DOPLER, FODA and Gears) seek to provide a solution. Optimisation approaches that seek to select features and extract products from feature models need to take account of these traceability links and constraints. Consider Figure 3 (but ignore the arrow labelled 'infer', which will be discussed later). From a feature model, such as the reduced Motorola feature model on the lefthand side of the figure, we can select features to produce the products (mobile phone instances) on the righthand side of the figure.

Search based feature model selection is isomorphic to the previously studied search based requirements selection problem [163]. Search based requirements selection seeks to find requirement selections, while respecting constraints. There are thus many parallels between work on Search Based Requirements Optimisation and Search Based SPL optimisation, as we shall show in the remainder of this section.

White et al. [153] introduce an approach they call Filtered Cartesian Flattening to select features from a feature model. They formulate feature selection as a constrained single objective formulation (subject to a budget constraint) and solve it using Branch and Bound with Linear Programming (BBLP). They formulate feature selection as a Multi-Dimensional, Multiple-Choice (MDMC) knapsack problem. Since MDMC knapsack is known to be NP-hard, they conclude that previous (exact) approaches may not scale to larger feature model instances. They introduce a heuristic to filter choices, thereby reducing the search space and evaluate on synthetic models of up to 5,000 features. They report that their heuristic scales, while suffering only approximately 7% loss of solution quality.

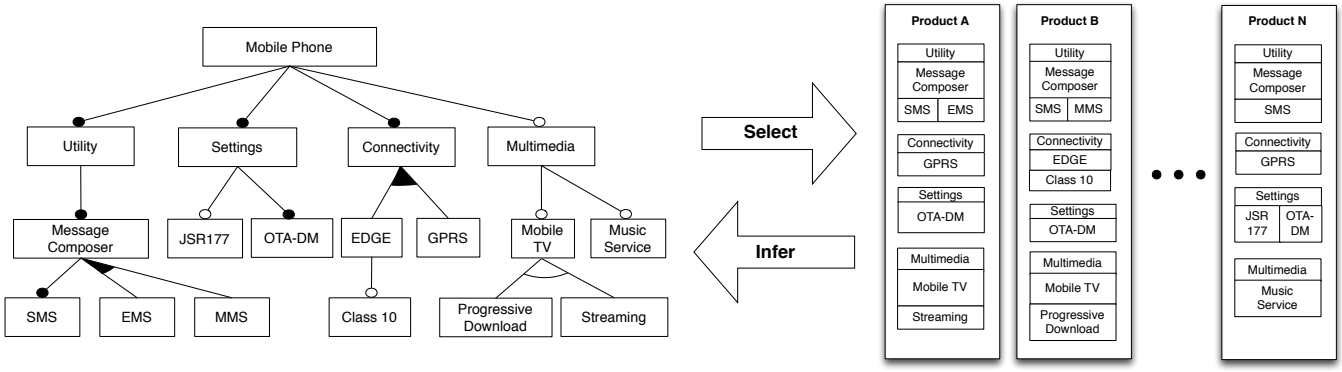

Figure 3: Selecting Features and Inferring Feature Models

Wu et al. [155] introduce (but do not evaluate) a framework of 6 typical reuse strategies and the problem of maximising product quality, subject to constraints (such as cost, failure rate and time). In so doing they reformulate the component selection and reuse problem [13] in terms of SPLs (subject to constraints). Component selection and reuse is, itself, a variation of the Next Release Problem [11] in Search Based Requirements Optimisation [162]. Therefore, this work makes an implicit link between previous work on search based software engineering for requirements selection and search based optimisation of choices pertaining to feature models. Wu et al. subsequently extended this earlier paper[2] [162] with a multi-objective optimisation formulation and a Mail Server System case study [154].

Bagheri et al. [10] propose an approach to the staged configuration of the product using a variant of the Analytic Hierarchy Process (AHP) that they term 'Stratified AHP'. This work is partly inspired by the use of AHP in requirements engineering optimisation, where it has a long history, dating back to Karlsson's foundational work [85] that was, itself, incorporated into the requirements management tool FocalPoint. AHP has the advantage that it minimises the number of questions required of the user, and can cater for conflicting responses about the choices to be made. This led to its use in the search based software engineering literature, not only in requirements, but also in 'human-in-the-loop' regression test optimisation [159].

In order to extract an instance from the feature model to create a specific product, the engineer has to answer a series of questions. The presence of constraints means that the answer to some questions may render certain subsequent questions inapplicable. Nohrer and Egyed [123] investigate iterative optimisation approaches to refine subsequent question choices, based on answers received. The goal was to reduce the number of overall questions required and to increase the amount of choice in the order in which questions are answered by the decision maker. It increases choice by identifying question subsets for which order is unimportant. Nohrer and Egyed report that the largest feature model with which they experimented involved an unoptimised sequence of 280 questions, indicating that optimisation is clearly important in this area. Chen and Erwig [24] also investigate the problem of optimising the sequence of questions posed to a decision maker, guided by a fitness function that measures feature selectivity.

[2]The earlier paper was in Chinese, while the latter is in English.

According Chen and Erwig's fitness function, a feature is more selective than another if choosing it causes more subsequent features to be automatically selected (according to the constraints pertinent to the feature model). Both Chen and Erwig and also Nohrer and Egyed used an iterative, greedy algorithm, which selects subsequent questions to be asked based on those previously encountered.

Guo et al. [96, 54] use a genetic algorithm to find SPL feature sets using a tool they implemented called GAFES. GAFES uses repair to transform invalid selections into valid ones, after crossover. Their fitness is value-per-unit-cost, so this is a single objective formulation from a technical search based optimisation perspective (they require a single fitness function). However, from the user's perspective, their approach combines cost and value objectives, just as, for example search based regression testing combines cost and value as value-per-unit-cost [39]. Guo et al. report that their GA approach out-performs the Filtered Cartesian Flattening approach [153] on synthetically generated feature models.

Muller [118] also formulate the choice of products to be built from an SPL as a cost-value trade off, using the simulated annealing computational search algorithm to find suggested choices of features that would form products that balance these trade offs. They focus on differing customer segments (stakeholder groups), observing that not all such groups can necessary be satisfied by the products offered (due to budgetary constraints).

Ognjanović et al. extract a business process model from a set of business process instances and then apply a genetic algorithm to search for business process configurations based on stakeholder requirements [126]. The contributions target the business community rather than the SPL community. Though the terminology is different to that used in both the SPL and the SBSE communities, the concepts and formulation as a search problem is very similar to that used in search based requirement optimisation [162] and SBSE for SPLs. This observation suggests additional synergies between SBSE for SPL, requirement and business process analysis.

Sayyad et al. provided a detailed investigation of the multi-objective feature selection problem in three related papers [137, 138, 140]. In their ICSE paper [140] they studied and evaluated metaheuristic algorithms for the multi-objective SPL feature selection problem. They make explicit the (often previously implicit) link between search based software engineering for requirements selection and search based optimisation of choices pertaining to feature models.

Their CMSBSE paper [138] extends the ICSE paper with a more detailed investigation of parameter tuning for crossover and mutation and explores the way different objectives become satisfied at different points in the optimisation process. This latter contribution may help the engineer to explain and account for optimisation decisions in their discussions with users. In their RESER paper [137] they replicate their previous findings from the ICSE paper that Indicator-Based Evolutionary Algorithm (IBEA) outperforms the well-known Non-Dominated Sorting Genetic Algorithm (NSGA-2). The replication shows that these findings also hold for the SPL feature selection problem, over multiple tuning choices.

While the formulation of Wu et al. [155], described earlier, was single objective, Sayyad et al. formulated a five-objective optimisation problem, in which the objectives were to minimise the total cost, defects, and violation and to maximise the total number of features offered by products, and the number of features reused from previous products. In so doing, Sayyad et al. transform the multi-objective next release problem [164] into a multi-objective software product line engineering problem.

Sayyad et al. [139] subsequently introduce heuristics to improve the performance of the IBEA they previously used [138, 140]. They report that these heuristics allow the IBEA to find sound and optimum configurations of very large variability models in the presence of multiple competing objectives.

Cruz et al. [36] use a hybrid approach, which combines fuzzy inference systems and the well-known multi-objective genetic algorithm, NSGAII, to help decision makers manage product lines by generating portfolios of products. These portfolios are based on user segments and the development cost of SPL products. Zhang et al. [161] proposed a fuzzy multi-objective approach to extend previous multi-objective SPL selection, but do not evaluate it.

Since White's work in 2008 there has been a departure from exact optimisation approaches for feature selection, to explore metaheuristics. Velazco's masters thesis [148] presents a comparison of exact and metaheuristic multi-objective approaches to feature selection. Velazco compares the SMT-based exact incremental approach called the Guided Improvement Algorithm with IBEA (which was used by Sayyad et al. [137, 138, 139, 140]). While the exact technique can find optimal solutions for small models (with 45 features) in two hours, IBEA scales to larger models with 290 features, which it covers sub-optimally in 20 minutes. Velazco's findings also indicate that significant effort is required to find suitable IBEA parameter tunings.

Wang and Pang [151] use Ant Colony Optimisation to optimise SPL feature selection, compering it to the Filtered Cartesian Flattening algorithm of White et al. [153] and the GAFES tool of Guo et al. [54, 96]. They report that their solution quality is, on average, 6% worse than Filtered Cartesian Flattening (though faster), while it is 10% better than GAFES (though slower). As such, it offers a modest compromise between these two existing solutions.

2.2 Search Based Feature Model Construction

One important problem in SPL engineering is the inference of products from a set of examples; an NP-hard problem [5]. This is somewhat analogous to the problems associated with requirements elicitation, in which the requirements are 'extracted' from user dialog or desired use-case scenarios.

However, whereas traditional requirement elicitation is primarily concerned with elicitation from users for new systems and next releases, for SPL engineering, there may already exist several instances of a product prior to the realisation that an SPL is required. Thus, for SPL feature model elicitation, there may often be a need to reverse engineer (or infer) a model from a set of instances. This problem is also illustrated in Figure 3, in which the arrow labelled 'infer' indicates that, from a set of products (on the righthand side) we can infer a feature model (on the lefthand side).

Chan et al. were among the first to address this problem using SBSE [22]. They use a genetic programming approach to generate customer satisfaction models and their relationship to design attributes, illustrating with the application of their approach to a digital camera SPL case study. In the same year, 2009, the connection between feature models and genetics was also discussed by Dhungana and Groher [37], though they did not use SBSE to exploit this link.

Linsbauer et al. [98] seek to learn feature models from instances using genetic programming, extending the previous work of Lopez-Herrejon et al. [68, 107], who used a genetic algorithm with post-crossover repair.

Many approaches to software product line engineering focus on feature models. Therefore, there is a requirement for thorough evaluation of the proposed tools and techniques using suitable feature models. A thorough evaluation of any search based software engineering approach requires both an experimental and an empirical dimension [57, 114].

In the experimental evaluation, feature model generators should be used in order to control the model's characteristics, thereby exploring the behaviour of the evaluated technique under laboratory controlled conditions. In the empirical evaluation, real-world feature models should be used in order to provide some indication of the technique's usefulness (and likely behaviour) on realistic models.

If there are sufficiently many real-world instances available, then the empirical evaluation may also play the role of the experimental evaluation. However, in many situations experimental evaluation can only be fully controlled using instance generators.

Segura et al. [142] address the need for instance generators for SPL evaluations. They model the problem of finding computationally hard feature models as an optimisation problem, which they address using an evolutionary algorithm. In this work, the goal is to use SBSE to search for feature models, with given size and characteristics, that are 'hard' in the sense that they drive the feature model analysis tool to consume time or memory.

The approach, implemented as a feature model generation tool called ETHOM, thus takes a tool and tunes the feature model 'parameters' to be a specifically hard case for the feature model analysis tool in hand. In this way, the ETHOM tool of Segura et al. is a search based 'stress test', akin to Briand et al.'s real-time stress tester [21]. It also shares a similar motivation and approach to the search based approach to tuning for more rigorous experimental evaluations of clone detect tools [150].

2.3 Search Based Architectural Improvement

Software architecture is a critical bridge between requirements and implementations [50]. For SPL engineering, the feature model plays a critical role in capturing and expressing relationships between the requirements for products.

It is often attractive and advantageous to interpose a product line architecture in-between the feature model and the products constructed from it. The product line architecture captures implementation concerns, with traceability links to the feature model and products. A product line architecture can be used to capture the core salient features, shared by all products on the product line. It also facilitates architectural exploration of the different variants and possible products that can be implemented.

There is a natural search-based problem of improving and repairing product line architectures to make them better fit their feature models and customer needs. There is also the related improvement and repair problem associated with the feature model itself.

Lopez-Herrejon and Egyed [101] suggest that SBSE could be used to search for inconsistencies in software product line feature models and architectures, a concept being developed within the SBSE4VM project on techniques to reverse engineer, evolve, and fix inconsistencies in systems with variability [102].

Colanzi and Vergilio [30, 31, 32, 33, 34] formulate Product Line Architecture (PLA) optimisation as a multi-objective SBSE problem, focusing on architectural objectives such as extensibility and modularity. Guizzo et al. [52, 53] extend this work by using design patterns in the optimisation process. Fung et al. [49] also mine rules to guide a genetic algorithm to search for design attribute settings, evaluating on a mobile phone case study.

Rezaul and Ruhe [83] and Karimpour and Ruhe [84] develop Ruhe et al.'s work [135] on SBSE for Requirements to support requirement theme analysis to identify and add new features to existing models. Themes could be regarded as components or cross-cutting SPL concerns.

2.4 Search Based SPL Testing

Approximately 50% of all research output on SBSE concerns problems related to software testing [65]. Testing problems naturally accommodate a search-based solution, because most test objectives can be captured by numerical assessment of test adequacy. These test adequacy criteria make good fitness function candidates.

In 2011 Engström and Runeson [40] and Neto et al. [120] both provided a systematic mapping of SPL testing, while in 2012 Lee et al. [95] provided a general survey of SPL testing. In this section, we focus on work on testing SPLs using computational search algorithms, thereby covering that part of the literature concerned with Search Based SPL Testing.

A natural choice in any approach to testing a software product line (with its characteristic feature combinations) is the use of Combinatorial Interaction Testing (CIT) [122]. In CIT, the goal is to form a test suite that exercises all t-way interactions of features in a feature model, for some choice of t (known as the 'strength' of the interactions tested). As t increases, the number of test cases required tends to grow exponentially. However, recent research [130] showed that higher strength testing can be practical in the presence of constraints. Fortunately, there is also evidence to suggest that the feature models used in software product line engineering are typically subject to multiple constraints [17].

The first author to explicitly suggest CIT for SPL was McGregor [111], whose proposal was subsequently extended by Cohen et al. [29], though the concept of CIT itself dates back to Mandl's work on compiler testing [109].

Perrouin et al. [128, 129] also use CIT to test feature choices in software product lines. They use a constraint satisfaction approach, implemented using the tool Alloy, and evaluated on the transactional SPL AspectOPTIMA.

Subsequently, Henard et al. [70, 71] proposed a search based approach to generate and prioritise combinatorial tests for SPLs, guided by a similarity measure. Ensan et al. [41] also use a genetic algorithm to search for SPL feature interactions. Kattepur et al. [86] use CIT for pairwise SPL testing, evaluating its effectiveness on several attributes of a crisis management case study.

Henard et al. [75] introduce an automated search-based process to test and fix feature models so that they adequately represent actual products. They propose [72] two mutation operators to derive erroneous feature models (mutants) from an original feature model. Henard et al. [73] also propose a genetic algorithm to handle multiple conflicting objectives in SPL test data generation.

Wang et al. [149] use a weighted GA to minimise SPL test suites, while seeking to retain fault detecting power, while Haslinger et al. [69] adapted a version of CASA [51], a well-known search based CIT tool, to improve its performance for SPL testing. Haslinger et al. report a speed up of over 60% on 133 publicly available feature models, while preserving the coverage of the generated tests.

Lopez-Herrejon et al. [105] propose a benchmark set of 19 different feature models for use in comparing CIT testing techniques and tools. They apply three different techniques in order to analyse their comparison framework and benchmark suite (and to evaluate these algorithms). The algorithms compared include CASA. Johansen et al. [80] also compare their proposed SPL CIT-based testing approach to CASA.

Xu et al. [156] extend the notion of test suite augmentation [136, 158] to SPLs. As the components of the product line are integrated to create products, the existing regression test suite has to be augmented to test the new products. Typically, a new product is an extension (or augmentation) of an existing product, expressed as a branch in the product line architecture. Therefore, it is likely that the existing test suite, can be augmented to cover the new product so-created.

Furthermore, existing test cases, which cover closely related products, are likely to form good seeds for a computational search for new test cases that cover the new product. Xu et al. use a genetic algorithm to augment the existing test suite in just this way, and also prioritise the order in which components are integrated into products to facilitate better test augmentation.

Henard et al. [74] also prioritise product configurations, seeking to maximise the number of feature interactions covered by a test suite. They also contributed an open source CIT tool for prioritised SPL testing called PLEDGE. Xu et al. and Henard et al. both published in the same year (2013).

Like Xu et al., Henard also seek to prioritise the order in which product configurations are considered. However, the prioritisation goals are different in each of the two approaches: Xu et al. seek to maximise productive reuse of existing test cases in the generation of new test cases, while Henard et al. seek to maximise feature interactions. Henard et al. also seek to maximise the feature interaction coverage for a given test suite (without changing it), while Xu et al. seek to augment test suites.

Lopez-Herrejon formulate the bi-objective problem of pairwise coverage and test suite minimisation as an exact optimisation problem [100]. For pairwise coverage (widely studied in CIT testing), all four combinations of inclusion and non-inclusion of each pair of feature values must be covered. They encode this coverage criterion as a maximisation problem (subject to constraints) so that a linear programming solution can be applied. The linear programming problem is to find the maximum coverage that can be achieved with n test cases.

By considering different values of n, $n > 0$, a Pareto front can be constructed which shows the trade-off between the number of test cases and the pairwise coverage achieved. Since Lopez-Herrejon et al. use linear programming for each value of n, the Pareto front thus-obtained is the exact global optimum for the software product line. They evaluate on 118 feature models taken from the SPL Conqueror Suite, ranging in size from 16 to 640 features.

The performance of many search based algorithms can be improved by seeding solutions known to be reasonable. In a global search, such as genetic algorithm, this can be done by seeding the initial population. In a local search, such as a hill climb, the starting point can also be seeded. Such seeding strategies have proved to be very effective in search based software engineering for modularisation [108], genetic improvement [8] and testing [4, 46]. Lopez-Herrejon et al. [103] compare seeding strategies for combinatorial testing of software product lines.

One alternative to CIT for feature models can be found in mutation testing [79]. Whereas CIT focuses on the interactions between different choices of features, mutation testing approaches focus on finding faults in feature models by generating test cases that can detect simulated faults in these models (mutated feature models).

Historically, mutation testing was used to assess test suite quality; suites that detect many mutants are good at finding simulated faults and, therefore, we hope that they will also reveal many real faults [79, 81]. However, more recently, research on mutation testing has been extended to generate test cases that are good at detecting mutants [47, 58].

Henard et al. [76] formulate feature models as boolean formulæ from which they generate program mutants to support SPL testing using mutation testing. Mutated feature models are used to test the software product line. Henard et al. compare their approach with random testing on 10 software product lines. They show that their approach finds more mutation faults and is better at minimising the number of tests required to find faults, when compared to random testing, a widely used baseline sanity check for SBSE [66].

More work is required to compare mutation and CIT based approaches to software product line testing. It seems likely that each approach will be good at finding different classes of faults and, therefore, this suggests the possibility of hybrid combinatorial-and-mutation approaches.

Furthermore, one of the advantages of mutation testing is that it is known to be adaptable; mutation approaches can easily simulate other test adequacy criteria [79]. Therefore, an interesting avenue for future work concerns mutation-based SPL testing approaches that simulate other SPL testing approaches, including combinatorial testing approaches. This might allow a single mutation based framework to be used for SPL testing, incorporating the advantages of many other approaches.

Mutation testing suffers from the problem of equivalent mutants [67]. That is, mutants are generated syntactically by altering the source code, but this may not have any semantic effect, leading to a mutant that is syntactically different, yet semantically equivalent, to the original program from which it is constructed. The problem is particularly pernicious because mutant equivalence is, in general, undecidable (for *program* mutants). However, since feature models are comparatively simple (generally loop free), we may hope for future work on formulations in which the mutant equivalence problem becomes decidable because the feature model language is sub-Turing complete. Such future work would imbue SPL mutation testing with advantages that remain unavailable to program mutation testing.

Filho et al. [44] use a random search to search an SPL counter example space. The counter examples of interest are constructed for a model based formulation of SPL design, in which a model separates the concerns of variability modelling from the 'realisation layer' (that maps variations to product choices). For such models, Filho et al. seek counter examples that yield invalid product models, despite being derived using valid variability models.

Filho et al. report results to indicate the scalability of their approach (which will come as no surprise to a search based software engineer, since random search is typically a very scalable technique [66]). However, though random search is typically scalable to large systems, it does not usually produce the highest quality solutions, nor often the greatest number of valid solutions.

Filho et al.'s work could be extended by future work on the formulation of suitable SBSE fitness functions that capture other important properties of a counter example, such as feature interactions, number of features and so on. In this way, model counter example generation could be formulated as a multi-objective search based test generation [61, 91, 132], selection [56, 157] or prioritisation [97] problem.

3. FUTURE WORK ON SBSE FOR SPL

In this section we present some possibilities for new avenues of work in which there community might apply computational search to software product line engineering.

3.1 AutoGrowing New SPL Product Branches

There has been tremendous progress in traditional Genetic Programming, which grows programs and functions from scratch [87, 133]. However, from the software engineering perspective, the task of growing an entire software system from scratch is rather akin to seeking to evolve human beings from bacteria; while theoretically possible, those seeking the practical application of genetic programming to this task may face a considerable wait.

In seeking to adapt and incorporate the successes of genetic programming in software engineering, search based software engineers have turned to a technique that has come to be known as 'genetic improvement' (but is also known as 'evolutionary improvement' and 'program improvement') [8, 9, 63, 90, 93, 127, 146, 152]. Genetic improvement is a development of genetic programming that seeks to improve existing programs, rather than evolving (or 'growing') new systems from scratch. As such, genetic improvement attacks a demanding (though significantly less challenging) problem that is more akin to evolving human beings from apes rather than from bacteria.

Genetic improvement can be thought of as an approach to constructing a set of intelligent generative programming [15] techniques. It can also be viewed as one potential solution approach to the long-standing challenge of program synthesis, towards which the wider computer science community has been optimistically (perhaps quixotically) heading for considerable time [110, 145].

Siegmund et al. [143] highlight the issue of non-functional properties such as performance, footprint size and energy consumption and the tensions between them; a choice of SPL product may have to meet many competing and conflicting non-functional properties. They introduce a system called 'SPL Conqueror' for capturing, measuring and choosing among non-functional properties.

Genetic improvement can be used to provide a 'Pareto surface' of programs [63]. This Pareto program surface contains a large number of different programs ('products' in SPL nomenclature), each of which share the same functionality, yet all of which differ in their non-functional properties.

The surface is constructed along several dimensions, each of which represents optimisation according to a specific non-functional property. All programs on the surface are Pareto optimal; none of them is superior to any of the others according to *all* of the non-functional properties. As such, the Pareto program surface represents a design space of programs in which the design dimensions concerned are non-functional product properties.

The concept of a 'Pareto front of SPL products' is not new to the SPL community. For example, Murashkin et al. [119] introduce a tool to visualise the Pareto front of optimal SPL variants according to cost, as well as non-functional properties such as energy use and security aspects. However, while existing work on Pareto fronts in the SPL community has concerned visualisation and analysis of the front, genetic improvement may offer a way to automatically *construct* the front or, more ambitiously, a surface (for multiple objectives).

In particular, recent results on genetic improvement have demonstrated that it is possible to evolve new versions of the system to optimise non-functional properties, while retaining the functional properties [89, 92, 131, 144, 152]. This approach could be a promising direction for automatically generating new branches of the product line according to differences in products that purely pertain to the non-functional behaviour of each product.

It seems unnecessarily labour-intensive to require human programmers to devote time and effort to the tedious task of constructing different versions of the same system for different platforms with different non-functional operating requirements. Surely we should be seeking ways to automatically grow (to 'AutoGrow') and optimise new products that differ simply on non-functional properties.

We may even be able to AutoGrow some new functionality: Grow and Graft Genetic Improvement (GGGI) [62] is a recent approach to genetic improvement in which new features are semi-automatically grown using genetic programming in isolation from the system into which they are subsequently grafted using search based software engineering. This is an instance of software transplantation [64], in which code from a donor program is transplanted into a host using SBSE. In this case, the donor of the transplantation process is not an existing system, but a piece of code that is grown in isolation.

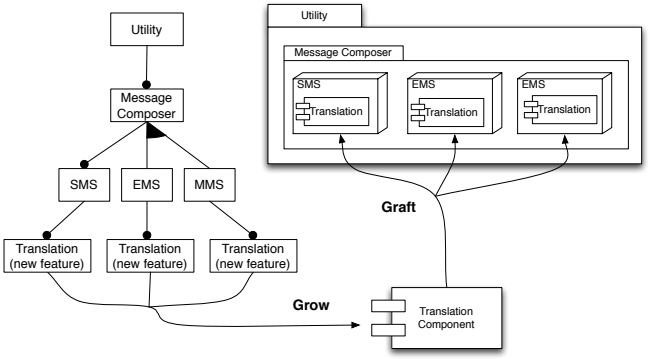

Figure 4: Growing and Grafting New SPL Features

In this 'grow' phase, the small fragment of code to be grown is evolved using genetic programming, guided by hints and suggestions from the programmer. These hints augment the more traditional (test-based) genetic programming fitness function. This code fragment so-grown (the donor) is subsequently grafted into an existing system (the host), by searching for a suitable insertion point and interface that connects the name spaces of donor and host. While software transplantation (in general) is analogous to biological organ transplantation, the GGGI approach is analogous to the proposed medical development of *in vitro* organ growth from stem cells for subsequent transplantation.

Perhaps more adventurous future work might investigate the degree to which genuinely new functionality could be grown as a branch of an existing product line using the GGGI approach. GGGI has been used to grow a natural language translation feature and graft it into the 200KLoC Pidgin Instant Messaging system [62]. This recent result indicates that future work could use GGGI to automatically grow similar new features and graft them into each product that needs the new feature (see Figure 4).

Such an approach would not necessarily lead to automated production code construction. Rather, the growth of the product line branch may remain the human programmer's responsibility, with GGGI providing valuable decision support. For example, a software engineer may be relieved to know that they could potentially 'autogrow' a customer's desired new feature. They may not ultimately deploy such autogrown code, but this knowledge may be useful in negotiations. Where such a branch cannot be easily autogrown, this may also provide valuable decision support; suggesting caution in the promises offered to the customer, because an unknown quantity of human effort will be required.

In a prototyping phase, developers may also speculatively grow and graft new branches to investigate the possibilities in the space of all possible products. Since these automatically grown extensions are essentially disposable (and therefore almost cost free), they provide programmers and their managers with ways to cheaply explore sub-regions of the product design space.

3.2 AutoMerging to Tackle Branchmania

It is easy for a feature model to grow alarmingly, leaving the developer with and unmanageably large number of different product variants. This situation has been termed 'branchmania' [18], because the product line architecture contains so many different branches that it becomes impossible to maintain and evolve a cost-effective product line.

One solution to the problem is to try and recognise when a product line is spiralling out of control and thereby take steps to intervene. A more radical solution is to slash and burn; in cases of branchmania perhaps the only obvious solution is to reduce the number of branches.

Unfortunately, neither of these solutions is ideal. Seeking to intervene raises all of the usual software engineering concerns of how and when to intervene and how to reconcile business and technical goals. Simply removing branches may provide a temporary amelioration, yet simultaneously reduce longer-term customer satisfaction, without even guaranteeing the absence of future branchmania.

SBSE and related techniques can be used to extract implicit parameters, making them tuneable [20, 59, 77]. Computational search can also tune [99, 150] parameters that may be already explicit, or otherwise rendered explicit.

Future work could therefore develop techniques to control branchmania by identifying branch similarities, extracting parameters and subsequently searching for suitable tunings that yield individual products. A set of child branches of a shared parent could thereby be merged into a single modified parent with an additional set of parameters that capture the variability previously present in the children. In this way, a combination of parameter extraction and tuned parameter instantiation could merge several products into a single parameterised product. We would be migrating apparently varying features into a parent branch parameter.

3.3 Retargetting Existing SBSE at SPL

The previous subsection focussed exclusively on genetic programming for SPL engineering. We suggested approaches in which the products themselves are automatically improved using techniques extended from recent advances in genetic improvement. In the next subsection we broaden our focus to consider SBSE techniques, more generally, that might find application in SPL engineering. We suggest ways in which other computational search algorithms can be used to support SPL planing, testing and scalability, without directly modifying the products themselves.

SPL Project Planning

There has been a great deal of work on search based approaches to software project management. Problems of the assignment of staff to projects [14, 25], management of risk [7, 42], and project scheduling [6, 26] have all been addressed in the literature.

Chastek and McGregor [23] suggest that careful project planning is even more important for software product lines than for physical product lines, such as 'consumer white goods' and automobiles. Results from search based software project management may therefore find further applications in software product line project planning.

Searching for Model Counter Examples

Classen et al. [28] apply model checking to SPLs, to check properties of all products generated from the SPL, while Cordy et al. [35] use model checking to search for SPL counter examples. Search based techniques such as Ant Colony Optimisation have proved effective at finding counter examples in other large model checking state spaces [3]. Formulations of software product line verification as model checking may therefore also benefit from search based software engineering.

Multicore Parallelisation for Scalability

Computational search algorithms are often referred to as 'embarrassingly parallel' because they naturally allow parallel formulations that can yield significant scalability enhancement. For instance, the fitness computation for each genetic algorithm individual can easily be computed in parallel. A more local search, such as a hill climbing algorithm, typically performs multiple restarts in a sequential setting, but all such restarts could also be performed in parallel.

Search based software engineers have realised the possibility of parallelisation as a route to scalability using computing clusters [108, 116] and, more recently, adapting General-Purpose Graphics Processing Units (GPGPUs) for search based regression test selection and prioritisation [97, 160] and for combinatorial interaction testing applied to SPLs [104]. Parallel SBSE may prove to be particularly important in scaling computational search algorithms to handle large-scale software product lines.

4. CONCLUSIONS

We attempted to comprehensively survey work on SBSE for SPL, an area for which we reveal a large recent upsurge in activity. Our literature search found 58 papers on SBSE for SPL, of which 52 were published since 2010. The most active area is SPL testing (22 papers), followed by SPL feature selection (18 papers), PLA improvement (12 papers) and SPL feature extraction (6 papers).

We also highlighted several recent advances in the use of genetic programming in SBSE research, explaining how these might address challenging problems for software product lines, such as search based branch merge (to reduce branchmania) and Grow and Graft Genetic Improvement (GGGI) to automatically augment a software product line with new branches.

5. ACKNOWLEDGEMENTS

We are grateful to the SPLC 2014 Organising Committee for the invitation to write this paper for the conference, and the invitation to Mark Harman to give the associated keynote talk on SBSE for SPL. Thanks also to Myra Cohen, Thelma Elita Colanzi Lopes, Roberto Erick Lopez-Herrejon, Tim Menzies, Mike Papadakis and Guenther Ruhe for their comments on an earlier draft of this paper.

Having sent the final draft for review by these authors we were made aware of the mapping study by Lopez-Herrejon et al. [106], which appeared online 2 days before our camera ready copy was due. Readers interested in our paper may also wish to consult this mapping study.

This work is supported by the Engineering and Physical Sciences Research Council (the EPSRC) grants Genetic Improvement of Software for Multiple Objectives (GISMO: EP/I033688) and Dynamic Adaptive Automated Software Engineering (DAASE: EP/J017515).

DAASE is an EPSRC 'programme grant' collaborative investment in SBSE in the UK. It is funded from June 2012 to May 2018, with matching support from EPSRC and University College London (UCL) and the Universities of Birmingham, Stirling and York. These four host universities will complement the 22 EPSRC-funded post doctoral researchers with 26 fully funded PhD studentships and 6 permanent faculty positions (assistant and associate professorships).

6. REFERENCES

[1] 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.

[2] 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.

[3] E. Alba and F. Chicano. ACOhg: Dealing with Huge Graphs. In *9th Conference on Genetic and Evolutionary Computation*, pages 10–17, July 2007.

[4] N. Alshahwan and M. Harman. Automated web application testing using search based software engineering. In *26th International Conference on Automated Software Engineering*, pages 3–12, Nov. 2011.

[5] N. Andersen, K. Czarnecki, S. She, and A. Wasowski. Efficient synthesis of feature models. In *16th International Software Product Line Conference*, pages 106–115, 2012.

[6] G. Antoniol, M. Di Penta, and M. Harman. The use of search-based optimization techniques to schedule and staff software projects: An approach and an empirical study. *Software – Practice and Experience*, 41(5):495–519, Apr. 2011.

[7] G. Antoniol, S. Gueorguiev, and M. Harman. Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In *ACM Genetic and Evolutionary Computation COnference (GECCO 2009)*, pages 1673–1680, Montreal, Canada, 8th – 12th July 2009.

[8] A. Arcuri, D. R. White, J. A. Clark, and X. Yao. Multi-objective improvement of software using co-evolution and smart seeding. In *7th International Conference on Simulated Evolution and Learning*, pages 61–70, Dec. 2008.

[9] A. Arcuri and X. Yao. A novel co-evolutionary approach to automatic software bug fixing. In *IEEE Congress on Evolutionary Computation*, pages 162–168, June 2008.

[10] E. Bagheri, M. Asadi, D. Gasevic, and S. Soltani. Stratified analytic hierarchy process: Prioritization and selection of software features. In *14th International Conference on Software Product Lines*, pages 300–315, Sept. 2010.

[11] A. Bagnall, V. Rayward-Smith, and I. Whittley. The next release problem. *Information and Software Technology*, 43(14):883–890, Dec. 2001.

[12] M. Bakal and C. W. Krueger. The Rhapsody/Gears bridge – SPL for MDD. In *11th International Conference on Software Product Lines*, pages 139–140, Sept. 2007.

[13] P. Baker, M. Harman, K. Steinhöfel, and A. Skaliotis. Search based approaches to component selection and prioritization for the next release problem. In *22nd International Conference on Software Maintenance*, pages 176–185, Sept. 2006.

[14] A. Barreto, M. de Oliveira Barros, and C. M. L. Werner. Staffing a software project: a constraint satisfaction and optimization-based approach. *Computers & Operations Research*, 35(10):3073–3089, Oct. 2008.

[15] B. Barth, G. Butler, K. Czarnecki, and U. W. Eisenecker. Generative programming. In *ECOOP Workshops*, pages 135–149, 2001.

[16] K. Berg, J. Bishop, and D. Muthig. Tracing software product line variability – from problem to solution space. In *Annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 182–191, 2005.

[17] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability modeling in the real: a perspective from the operating systems domain. In *25th International Conference on Automated Software Engineering*, pages 73–82, Sept. 2010.

[18] C. Bird and T. Zimmermann. Assessing the value of branches with what-if analysis. In *20th Symposium on the Foundations of Software Engineering*, pages 45(1–10), 2012.

[19] Y. Bontemps, P. Heymans, P.-Y. Schobbens, and J.-C. Trigaux. The semantics of FODA feature diagrams. In *Workshop on Software Variability Management for Product Derivation*, Aug. 2004.

[20] N. Brake, J. R. Cordy, E. Dancy, M. Litoiu, and V. Popescu. Automating discovery of software tuning parameters. In *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 65–72, 2008.

[21] L. C. Briand, Y. Labiche, and M. Shousha. Stress testing real-time systems with genetic algorithms. In *Genetic and Evolutionary Computation Conference*, pages 1021–1028, 2005.

[22] K. Y. Chan, C. K. Kwong, and T. C. Wong. Modelling customer satisfaction for product development using genetic programming. *Journal of Engineering Design*, 22(1):55–68, Jan. 2009.

[23] G. Chastek and J. D. McGregor. Production planning in a software product line organization. In *12th International Software Product Line Conference*, pages 369–369, Sept. 2008.

[24] S. Chen and M. Erwig. Optimizing the product derivation process. In *15th International Conference on Software Product Lines*, pages 35–44, Aug. 2011.

[25] W.-N. Chen and J. Zhang. Ant colony optimization for software project scheduling and staffing with an event-based scheduler. *IEEE Transactions on Software Engineering*, 39(1):1–17, Jan. 2013.

[26] F. Chicano, F. Luna, A. J. Nebro, and E. Alba. Using multi-objective metaheuristics to solve the software project scheduling problem. In *13th Annual Conference on Genetic and Evolutionary Computation*, pages 1915–1922, July 2011.

[27] A. Classen, Q. Boucher, and P. Heymans. A text-based approach to feature modelling: Syntax and semantics of TVL. *Science of Computer Programming*, 76(12):1130–1143, 2011.

[28] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. *IEEE Transactions on Software*

Engineering, 39(8):1069–1089, Aug. 2013.

[29] M. B. Cohen, M. B. Dwyer, and J. Shi. Coverage and adequacy in software product line testing. In *Workshop on the Role of Software Architecture for Testing and Analysis*, pages 53–63, July 2006.

[30] T. E. Colanzi. Search based design of software product lines architectures. In *34th International Conference on Software Engineering*, pages 1507–1510, 2012.

[31] T. E. Colanzi and S. R. Vergilio. Applying search based optimization to software product line architectures: Lessons learned. In *4th International Symposium on Search Based Software Engineering*, pages 259–266, Sept. 2012.

[32] T. E. Colanzi and S. R. Vergilio. Representation of software product line architectures for search-based design. In *1st International Workshop on Combining Modelling and Search-Based Software Engineering*, pages 28–33, 2013.

[33] T. E. Colanzi and S. R. Vergilio. A feature-driven crossover operator for product line architecture design optimization. In *38th International Computers, Software and Applications Conference*, Västeras, Sweden, 2014. To appear.

[34] T. E. Colanzi, S. R. Vergilio, I. M. S. G. S., and W. N. Oizumi. A search-based approach for software product line design. In *18th International Software Product Line Conference*, 2014. To appear.

[35] M. Cordy, P. Heymans, A. Legay, P.-Y. Schobbens, B. Dawagne, and M. Leucker. Counterexample guided abstraction refinement of product-line behavioural models. In *22nd International Symposium on the Foundations of Software Engineering*, 2014. To appear.

[36] J. Cruz, P. S. Neto, R. Britto, R. Rabelo, W. Ayala, T. Soares, and M. Mota. Toward a hybrid approach to generate software product line portfolios. In *IEEE Congress on Evolutionary Computation*, pages 2229–2236, 2013.

[37] D. Dhungana and I. Groher. Genetics as a role model for software variability management. In *31st International Conference on Software Engineering Companion Volume*, pages 239–242, May 2009.

[38] D. Dhungana, P. Grünbacher, and R. Rabiser. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, 18(1):77–114, 2011.

[39] S. Elbaum, A. Malishevsky, and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *23rd International Conference on Software Engineering*, pages 329–338, May 2001.

[40] E. Engström and P. Runeson. Software product line testing – a systematic mapping study. *Information & Software Technology*, 53(1):2–13, 2011.

[41] F. Ensan, E. Bagheri, and D. Gasevic. Evolutionary search-based test generation for software product line feature models. In *24th International Conference on Advanced Information Systems Engineering*, pages 613–628, 2012.

[42] F. Ferrucci, M. Harman, J. Ren, and F. Sarro. Not going to take this anymore: Multi-objective overtime planning for software engineering projects. In *35th International Conference on Software Engineering*, 2013.

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

[44] J. B. F. Filho, O. Barais, M. Acher, B. Baudry, and J. L. Noir. Generating counterexamples of model-based software product lines: an exploratory study. In *17th International Software Product Line Conference*, pages 72–81, Aug. 2013.

[45] G. Fraser and A. Arcuri. Evosuite: automatic test suite generation for object-oriented software. In *8th European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 416–419, Sept. 2011.

[46] G. Fraser and A. Arcuri. The seed is strong: Seeding strategies in search-based software testing. In *5th International Conference on Software Testing, Verification and Validation*, pages 121–130, Apr. 2012.

[47] G. Fraser and A. Zeller. Mutation-driven generation of unit tests and oracles. In *International Symposium on Software Testing and Analysis*, pages 147–158, 2010.

[48] F. G. Freitas and J. T. Souza. Ten years of search based software engineering: A bibliometric analysis. In *3rd International Symposium on Search based Software Engineering*, pages 18–32, Sept. 2011.

[49] C. K. Fung, C. Kwong, K. Y. Chan, and H. Jiang. A guided search genetic algorithm using mined rules for optimal affective product design. *Engineering Optimization*, 46(8):1094–1108, 2014.

[50] D. Garlan. Software architecture: A travelogue. In *Future of Software Engineering*, pages 29–39, 2014.

[51] B. J. Garvin, M. B. Cohen, and M. B. Dwyer. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering*, 16(1):61–102, 2011.

[52] G. Guizzo, T. E. Colanzi, and S. R. Vergilio. Applying design patterns in product line search-based design: Feasibility analysis and implementation aspects. In *Chilean Computing Conference*, 2013.

[53] G. Guizzo, T. E. Colanzi, and S. R. Vergilio. Applying design patterns in search-based product line architecture design. In *6th Symposium on Search Based Software Engineering*, Aug. 2014. To appear.

[54] J. Guo, J. White, G. Wang, J. Li, and Y. Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *The Journal of Systems and Software*, 84(12):2208–2221, Dec. 2011.

[55] M. Harman. How SBSE can support construction and analysis of predictive models (keynote). In *6th International Conference on Predictive Models in Software Engineering*, 2010.

[56] M. Harman. Making the case for MORTO: Multi objective regression test optimization (invited position paper). In *1st International Workshop on Regression Testing*, 2011.

[57] M. Harman, E. Burke, J. A. Clark, and X. Yao.

Dynamic adaptive search based software engineering (keynote paper). In *6th International Symposium on Empirical Software Engineering and Measurement*, pages 1–8, Sept. 2012.

[58] M. Harman, Y. Jia, and B. Langdon. Strong higher order mutation-based test data generation. In *8th European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 212–222, Sept. 2011.

[59] M. Harman, Y. Jia, W. B. Langdon, J. Petke, I. H. Moghadam, S. Yoo, and F. Wu. Genetic improvement for adaptive software engineering (keynote). In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 1–4, 2014.

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

[61] M. Harman, K. Lakhotia, and P. McMinn. A multi-objective approach to search-based test data generation. In *9th Annual Conference on Genetic and Evolutionary Computation*, pages 1098–1105, July 2007.

[62] M. Harman, W. B. Langdon, and Y. Jia. Babel pidgin: SBSE can grow and graft entirely new functionality into a real world system. In *6th Symposium on Search Based Software Engineering*, Aug. 2014. To appear.

[63] 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 *27th International Conference on Automated Software Engineering*, pages 1–14, Sept. 2012.

[64] M. Harman, W. B. Langdon, and W. Weimer. Genetic programming for reverse engineering (keynote paper). In *20th Working Conference on Reverse Engineering*, Oct. 2013.

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

[66] M. Harman, P. McMinn, J. Souza, and S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification: LASER 2009-2010*, pages 1–59. Springer, 2012.

[67] M. Harman, X. Yao, and Y. Jia. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *36th International Conference on Software Engineering*, June 2014. To appear.

[68] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. Reverse engineering feature models from programs' feature sets. In *18th Working Conference on Reverse Engineering*, pages 308–312, Oct. 2011.

[69] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. Improving CASA runtime performance by exploiting basic feature model analysis. *CoRR*, abs/1311.7313, 2013.

[70] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize T-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, 2014. To appear.

[71] C. Henard, M. Papadakis, G. Perrouin, J. Klein, P. Heymans, and Y. L. Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize T-wise test suites for large software product lines. *CoRR*, abs/1211.5451, 2012.

[72] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Assessing software product line testing via model-based mutation: An application to similarity testing. In *6th International Conference on Software Testing, Verification and Validation Workshops*, pages 188–197, 2013.

[73] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Multi-objective test generation for software product lines. In *17 International Software Product Line Conference*, pages 62–71, 2013.

[74] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Pledge: A product line editor and test generation tool. In *17th International Software Product Line Conference Co-located Workshops*, pages 126–129, 2013.

[75] C. Henard, M. Papadakis, G. Perrouin, J. Klein, and Y. L. Traon. Towards automated testing and fixing of re-engineered feature models. In *35th International Conference on Software Engineering*, pages 1245–1248, 2013.

[76] C. Henard, M. Papadakis, and Y. L. Traon. Mutation-based generation of software product line test con
figurations. In *6th Symposium on Search Based Software Engineering*, Aug. 2014. To appear.

[77] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 199–212, Mar. 2011.

[78] Y. Jia and M. Harman. Milu: A customizable, runtime-optimized higher order mutation testing tool for the full C language. In *3rd Testing Academia and Industry Conference – Practice and Research Techniques*, pages 94–98, Aug. 2008.

[79] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[80] M. F. Johansen, Ø. Haugen, and F. Fleurey. An algorithm for generating T-wise covering arrays from large feature models. In *16th International Software Product Line Conference*, pages 46–55, 2012.

[81] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *22nd International Symposium on the Foundations of Software Engineering (FSE 2014)*, 2014. To appear.

[82] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Carnegie-Mellon University,

Software Engineering Institute, 1990.

[83] M. R. Karim and G. Ruhe. Bi-objective genetic search for release planning in support of themes. In *6th Symposium on Search Based Software Engineering*, Aug. 2014. To appear.

[84] R. Karimpour and G. Ruhe. Bi-criteria genetic search for adding new features into an existing product line. In *1st International Workshop on Combining Modelling and Search-Based Software Engineering*, pages 34–38, May 2013.

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

[86] A. Kattepur, S. Sen, B. Baudry, A. Benveniste, and C. Jard. Variability modeling and QoS analysis of web services orchestrations. In *International Conference on Web Services*, pages 99–106, 2010.

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

[88] 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, Jan. 2013.

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

[90] W. B. Langdon and M. Harman. Optimising existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, 2014. To appear.

[91] W. B. Langdon, M. Harman, and Y. Jia. Efficient multi objective higher order mutation testing with genetic programming. *Journal of Systems and Software*, 83(12):2416–2430, 2010.

[92] W. B. Langdon, M. Modat, J. Petke, and M. Harman. Improving 3D medical image registration CUDA software with genetic programming. In *16th Conference on genetic and evolutionary computation conference (GECCO 2014)*, Vancouver, Canada, 12-15 July 2014. ACM. To Appear.

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

[94] 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.

[95] J. Lee, S. Kang, and D. Lee. A survey on software product line testing. In *16th International Software Product Line Conference – Volume 1*, pages 31–40, 2012.

[96] J. Li, X. Liu, Y. Wang, and J. Guo. Formalizing feature selection problem in software product lines using 0-1 programming. In *6th International Conference on Intelligent Systems and Knowledge Engineering*, 2011.

[97] Z. Li, Y. Bian, R. Zhao, and J. Cheng. A fine-grained parallel multi-objective test case prioritization on GPU. In *5th International Symposium on Search Based Software Engineering*, pages 111–125, Aug. 2013.

[98] L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. Feature model synthesis with genetic-programming. In *6th Symposium on Search Based Software Engineering*, Aug. 2014. To appear.

[99] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. Improving trace accuracy through data-driven configuration and composition of tracing features. In *9th joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 378–388, Aug. 2013.

[100] R. E. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, and E. Alba. Multi-objective optimal test suite computation for software product line pairwise testing. In *29th International conference on software maintenance*, pages 404–407, 2013.

[101] R. E. Lopez-Herrejon and A. Egyed. Searching the variability space to fix model inconsistencies: A preliminary assessment. In *3rd International Symposium on Search based Software Engineering*, Sept. 2011. Fast Abstract.

[102] R. E. Lopez-Herrejon and A. Egyed. SBSE4VM: Search based software engineering for variability management. In *17th European Conference on Software Maintenance and Reengineering*, pages 441–444, 2013.

[103] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba. Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines. In *IEEE Congress on Evolutionary Computation*, pages 387–396, July 2014.

[104] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. A parallel evolutionary algorithm for prioritized pairwise testing of software product lines. In *Genetic and Evolutionary Computation Conference*, 2014. To appear.

[105] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, E. N. Haslinger, A. Egyed, and E. Alba. Towards a benchmark and a comparison framework for combinatorial interaction testing of software product lines. *CoRR*, abs/1401.5367, 2014.

[106] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, L. Linsbauer, A. Egyed, and E. Alba. A hitchhiker's guide to search-based software engineering for software product lines. *CoRR*, abs/1406.2823, 2014.

[107] R. E. Lopez-Herrejon, J. A. Galindo, D. Benavides, S. Segura, and A. Egyed. Reverse engineering feature models with evolutionary algorithms: An exploratory study. In *4th International on Search Based Software Engineering – Symposium*, pages 168–182, Sept. 2012.

[108] K. Mahdavi, M. Harman, and R. M. Hierons. A multiple hill climbing approach to software module clustering. In *International Conference on Software Maintenance*, pages 315–324, Sept. 2003.

[109] R. Mandl. Orthogonal latin squares: an application of experiment design to compiler testing. *Communications of the ACM*, 28(10):1054–1058, Oct. 1985.

[110] Z. Manna and R. J. Waldinger. Toward automatic

program synthesis. *Communications of the ACM*, 14(3):151–164, 1971.

[111] J. McGregor. Testing a software product line. Technical Report CMU/SEI-2001-TR-022, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2001.

[112] P. McMinn. Search-based software testing: Past, present and future. In *International Workshop on Search-Based Software Testing*, pages 153–163, Mar. 2011.

[113] A. Metzger and K. Pohl. Software product line engineering and variability management: Achievements and challenges. In *Future of Software Engineering*, pages 70–84, 2014.

[114] B. Meyer, H. Gall, M. Harman, and G. Succi. Empirical answers to fundamental software engineering problems (panel paper). In *European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 14–18. ACM, Aug. 2013.

[115] 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.

[116] B. S. Mitchell, M. Traverso, and S. Mancoridis. An architecture for distributing the computation of software clustering algorithms. In *IEEE/IFIP Working Conference on Software Architecture*, pages 181–190, 2001.

[117] I. H. Moghadam and Mel Ó Cinnéide. Code-Imp: A tool for automated search-based refactoring. In *4th Workshop on Refactoring Tools*, pages 41–44, 2011.

[118] J. Muller. Value-based portfolio optimization for software product lines. In *15th International Software Product Line Conference*, pages 15–24, Aug. 2011.

[119] A. Murashkin, M. Antkiewicz, D. Rayside, and K. Czarnecki. Visualization and exploration of optimal variants in product line engineering. In *17th International Software Product Line Conference*, pages 111–115, 2013.

[120] P. A. S. Neto, I. d. Machado, J. D. McGregor, E. S. de Almeida, and S. R. d. Meira. A systematic mapping study of software product lines testing. *Information & Software Technology*, 53(5):407–423, 2011.

[121] 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, Aug. 2008.

[122] C. Nie and H. Leung. A survey of combinatorial testing. *ACM Computing Surveys*, 43(2):11:1–11:29, 2011.

[123] A. Nöhrer and A. Egyed. Optimizing user guidance during decision-making. In *15th International Conference on Software Product Lines*, pages 25–34, Aug. 2011.

[124] L. Northrop and P. Clements. *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[125] M. Ó Cinnéde, L. Tratt, M. Harman, S. Counsell, and I. H. Moghadam. Experimental assessment of software metrics using automated refactoring. In *6th International Symposium on Empirical Software Engineering and Measurement*, pages 49–58, Sept. 2012.

[126] I. Ognjanović, B. Mohabbati, D. Gaevic, E. Bagheri, and M. Bokovic. A metaheuristic approach for the configuration of business process families. In *International Conference on Services Computing*, pages 25–32, June 2012.

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

[128] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y. L. Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 20(3–4):605–643, 2012.

[129] G. Perrouin, S. Sen, J. Klein, B. Baudry, and Y. L. Traon. Automated and scalable T-wise test case generation strategies for software product lines. In *3rd International Conference on Software Testing, Verification and Validation*, pages 459–468, 2010.

[130] J. Petke, M. B. Cohen, M. Harman, and S. Yoo. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In *European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 26–36, Aug. 2013.

[131] J. Petke, M. Harman, W. B. Langdon, and W. Weimer. Using genetic improvement & code transplants to specialise a C++ program to a problem class. In 17^{th} *European Conference on Genetic Programming (EuroGP)*, Granada, Spain, April 2014. To Appear.

[132] G. H. L. Pinto and S. R. Vergilio. A multi-objective genetic algorithm to test data generation. In *22nd International Conference on Tools with Artificial Intelligence*, pages 129–134, 2010.

[133] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

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

[135] M. O. Saliu and G. Ruhe. Bi-objective release planning for evolving software systems. In *European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, pages 105–114. ACM, Sept. 2007.

[136] R. A. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J. Harrold. Test-suite augmentation for evolving software. In *23rd Automated Software Engineering*, pages 218–227, 2008.

[137] A. S. Sayyad, K. Goseva-Popstojanova, T. Menzies, and H. Ammar. On parameter tuning in search-based software engineering: A replicated empirical study. In *International Workshop on Replication in Empirical Software Engineering Research*, Oct. 2013.

[138] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Optimum feature selection in software product lines: Let your model and values guide your search. In *1st*

International Workshop on Combining Modelling and Search-Based Software Engineering, pages 22–27, 2013.

[139] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable product line configuration: A straw to break the camel's back. In *28th International Conference on Automated Software Engineering*, pages 465–474, 2013.

[140] A. S. Sayyad, T. Menzies, and H. Ammar. On the value of user preferences in search-based software engineering: a case study in software product lines. In *35th International Conference on Software Engineering*, pages 492–501, 2013.

[141] P. Schobbens, P. Heymans, and J.-C. Trigaux. Feature diagrams: A survey and a formal semantics. In *14th International Conference on Requirements Engineering*, pages 139–148, Sept. 2006.

[142] S. Segura, J. A. Parejo, R. M. Hierons, D. Benavides, and A. R. Cortés. Automated generation of computationally hard feature models using evolutionary algorithms. *Expert Systems Applications*, 41(8):3975–3992, 2014.

[143] N. Siegmund, M. Rosenmüller, M. Kuhlemann, C. Kästner, S. Apel, and G. Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3–4):487–517, 2012.

[144] 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.

[145] J. Stark and A. Ireland. Towards automatic imperative program synthesis through proof planning. In *Conference on Automated Software Engineering*, pages 44–51, 1999.

[146] J. Swan, M. G. Epitropakis, and J. R. Woodward. Gen-o-fix: An embeddable framework for dynamic adaptive genetic improvement programming. Technical Report CSM-195, Computing Science and Mathematics, University of Stirling, 2014.

[147] P. Tonella. Evolutionary testing of classes. In *International Symposium on Software Testing and Analysis*, pages 119–128, July 2004.

[148] R. E. O. Velazco. Comparison of exact and approximate multi-objective optimization for software product lines. Master's thesis, University of Waterloo, 2013.

[149] S. Wang, S. Ali, and A. Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *15th Annual Conference on Genetic and Evolutionary Computation*, pages 1493–1500, July 2013.

[150] T. Wang, M. Harman, Y. Jia, and J. Krinke. Searching for better configurations: a rigorous approach to clone evaluation. In *European Software Engineering Conference and the Symposium on the Foundations of Software Engineering*, pages 455–465, Aug. 2013.

[151] Y. Wang and J. Pang. Ant colony optimization for feature selection in software product lines. *Journal of Shanghai Jiaotong University (Science)*, 19(1):50–58, 2014.

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

[153] J. White, B. Doughtery, and D. C. Schmidt. Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints. In *12th International Conference on Software Product Lines*, pages 209–216, Sept. 2008.

[154] Z. Wu, J. Tang, C. K. Kwong, and C. Y. Chan. An optimization model for reuse scenario selection considering reliability and cost in software product line development. *International Journal of Information Technology & Decision Making*, 10(5):811–841, 2011.

[155] Z. Wu, J. Tang, and L. Wang. An optimization framework for reuse component selection in software product line. In *Control and Decision Conference*, pages 1880–1884, June 2009. In Mandarin Chinese.

[156] Z. Xu, M. B. Cohen, W. Motycka, and G. Rothermel. Continuous test suite augmentation in software product lines. In *17th International Software Product Line Conference*, pages 52–61, Aug. 2013.

[157] S. Yoo and M. Harman. Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software*, 83(4):689–701, 2010.

[158] 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.

[159] S. Yoo, M. Harman, P. Tonella, and A. Susi. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *International Conference on Software Testing and Analysis*, pages 201–212, July 2009.

[160] S. Yoo, M. Harman, and S. Ur. GPGPU test suite minimisation: search based software engineering performance improvement using graphics cards. *Journal of Empirical Software Engineering*, 18(3):550–593, June 2013.

[161] H. Zhang, R. Lin, H. Zou, F. Yang, and Y. Zhao. The collaborative configuration of service-oriented product lines based on evolutionary approach. In *International Conference on Services Computing*, pages 751–752, 2013.

[162] Y. Zhang, A. Finkelstein, and M. Harman. Search based requirements optimisation: Existing work and challenges. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 88–94, 2008.

[163] Y. Zhang, M. Harman, and S. L. Lim. Empirical evaluation of search based requirements interaction management. *Journal of Information and Software Technology*, 55(1):126–152, Jan. 2013.

[164] Y. Zhang, M. Harman, and A. Mansouri. The multi-objective next release problem. In *9th Annual Conference on Genetic and evolutionary computation*, pages 1129–1137, July 2007.