Dynamic Adaptive Search Based Software Engineering Needs Fast Approximate Metrics

Mark Harman*, John Clark† and Mel Ó Cinnéide‡
*University College London, CREST centre, UK
†University of York, UK.
‡University College Dublin, Ireland.

***Abstract*—Search Based Software Engineering (SBSE) uses fitness functions to guide an automated search for solutions to challenging software engineering problems. The fitness function is a form of software metric, so there is a natural and close interrelationship between software metics and SBSE. SBSE can be used as a way to experimentally validate metrics, revealing startling conflicts between metrics that purport to measure the same software attributes. SBSE also requires new forms of surrogate metrics. This topic is less well studied and, therefore, remains an interesting open problem for future work. This paper[1] overviews recent results on SBSE for experimental metric validation and discusses the open problem of fast approximate surrogate metrics for dynamic adaptive SBSE.**

I. Introduction

Search Based Software Engineering (SBSE) is concerned with the development of techniques, based on computational search, to solve hard problems in software engineering. The term 'search based software engineering' was introduced in 2001 [1]. Since then there has been an explosion of activity in this area. Many different computational search algorithms have been used including genetic algorithms, simulated annealing, hill climbing, and genetic programming.

SBSE has been applied to a wide variety of software engineering problems spanning the entire spectrum of activities that can be broadly characterised as software engineering, including requirements analysis, project management, software design and redesign, coding and implementation, testing, bug fixing, maintenance, reengineering and refactoring [2].

SBSE involves reformulating software engineering problems as search problems [3]. This reformulation requires a representation of the problem to be solved and a fitness function with which we can measure progress towards the achievement of the overall software engineering objective [4]. Naturally, as software engineers, we have many different candidate representations for our problems, and usually a number of these are amenable to the SBSE approach. Also, because of the rich and vibrant research of the software metrics and measurement community, the would-be search based software engineer does not have far to look when selecting a fitness function: In 2004, Harman and Clark argued that 'metrics are fitness functions too' [5]. In this paper, we seek to develop this agenda of metrics as fitness functions.

The 2004 paper [5] focused largely on the observation that search based software engineering could be used as a mechanism for investigating the validity of software metrics. The goals of this research programme have been further developed since 2004, notably in work on search based transformation and refactoring [6], [7], [8], [9], [10].

Recently, Ó Cinnéide et al. [11] demonstrated a practical system that exploits search based refactoring as a means of dynamically validating software metrics.

In this position paper we overview the work in this research programme. This strand of work can be characterised as 'SBSE for metrics'. We also consider the other side of this coin, showing that the metrics community has a lot to offer the SBSE research agenda: 'Metrics for SBSE'.

Specifically, we will explain why we believe a new form of metrics research is required to develop surrogate fitness functions for the emergent paradigm of Dynamic Adaptive Search Based Software Engineering [12]. This is a comparatively less well studied topic and one that the authors believe deserves greater attention.

II. Metrics Are Fitness Functions Too

In a recent paper at the 6^{th} ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), we (Ó Cinnéide et al. [11]) developed the idea of metrics as fitness functions, specifically using search based refactoring as a means to evaluate metrics. This work was a development of an idea first put forward in 2004 [5], but which remained unimplemented, and therefore unevaluated, until the 2012 ESEM paper. This section presents an overview of this approach and some of our recent findings.

Our approach to metric validation uses SBSE to search for sequences of refactorings, experimentally examining the degree of agreement and disagreement between metrics' assessments of the sequence of refactored versions of the code. We do not evaluate a metric in isolation. Rather, we compare the effect of one metric (when used as a fitness function) with the effects on other metrics. Our search process only applies a refactoring if it improves at least one of the metrics under consideration. The other metrics may improve, disimprove or remain the same. This allows us to explore the relationship between different metrics.

If the metrics are measuring the *same* thing, then should they not tend to agree on the impact of refactorings? In a perfect world, the answer to this question would always be 'yes'. However, metrics can never reside in a perfect world; there are too many imponderables. We used SBSE to explore just how imperfect is the world of metrics, focussing specifically on cohesion metrics. Our results revealed a startling level of disagreement between metrics, all of which claim to measure cohesion.

[1]This short paper is an invited paper, written to accompany the keynote given by Mark Harman at the 4^{th} International Workshop on Emerging Trends in Software Metrics (WeTSOM 2013). It overviews joint work with John Clark and Mel Ó Cinnéide on metrics as fitness functions, SBSE for experimental metric validation and Dynamic Adaptive SBSE.

Our previous work demonstrated the way in which SBSE can be used to assess metric agreement in a rigorous experimental setting and to pinpoint areas of weakness and divergence of outcomes. Our approach provides a way to experimentally assess metrics using a relative form of the standard foundational 'representation condition' of software measurement [13]. That is, we cannot easily assess how well metrics conform to the judgement of human experts (the *absolute* form of the representation condition), but we *can* assess their *relative* conformance to each other using Search Based Refactoring.

We applied our approach to five popular cohesion metrics, evaluating them on eight real-world Java systems consisting of approximately 300KLoC. The metrics studied were five cohesion metrics: Tight Class Cohesion (TCC) [14], Lack of Cohesion between Methods (LCOM5) [15], Class Cohesion (CC) [16], Sensitive Class Cohesion (SCOM) [17] and Low-level Similarity Base Class Cohesion (LSCC) [18].

In order to experimentally evaluate metrics using SBSE, we introduced 'metametrics' to measure the performance of metrics. These metametrics are 'metrics that measure metrics'. We believe many such metametrics might be defined and that this may be a profitable avenue for future research in software metrics and, in particular, for metrics validation. In this brief overview, we present some results from our previous study of one such metametric: volatility.

As defined in our ESEM paper

> "A volatile metric is one that is changed often by refactorings, whereas an inert metric is one that is changed infrequently by refactorings".

Volatility is a measure of how sensitive (or fine-grained) a metric is. Volatility may have a number of applications and ramifications. For example, should a metric be found to be volatile, then a software developer may not be so concerned by minor differences in metric outcomes between different software systems. By contrast, large changes in a comparatively inert metric between two successive releases of a system would be more worrying.

Table I (taken from our ESEM 2012 paper) shows the volatility of the 5 metrics we studied, averaged across all systems. Our results revealed that LSCC, CC and LCOM5 are all highly volatile metrics. Almost all (in fact 99%) of the refactorings applied affect them. TCC is notably different. The TCC metric displays far less volatility (in general) than LSCC, CC and LCOM5.

Therefore, in general, perhaps developers concerned about cohesion should take more notice of relatively small fluctuations in TCC between successive releases than similar fluctuations observed for LSCC, CC and LCOM5. In this way, our experimental metric validation can be useful to researchers investigating the properties of metrics and their performance and also to developers, guiding their interpretation of the significance of changes in metrics' values.

Our results indicate that volatility is not merely a property of metrics, but, as one might expect, it is a product of metric and system measured. This observation can be useful in practice: a developer can use our approach to pre-determine whether a metric is volatile for the application on which they are working, and this can inform decisions made about the outcome of subsequent measurements of the application.

	LSCC	TCC	SCOM	CC
TCC	0.60			
SCOM	0.70	0.58		
CC	0.10	0.01	-0.28	
LCOM5	-0.17	-0.21	-0.46	0.72

TABLE II
SPEARMAN RANK CORRELATION COEFFICIENTS BETWEEN EACH PAIR OF METRICS ACROSS ALL REFACTORINGS AND ALL APPLICATIONS. NOTICE THAT LCOM5 MEASURES *lack* OF COHESION, SO A NEGATIVE CORRELATION COEFFICIENT INDICATES A POSITIVE CORRELATION. THESE RESULTS TAKEN FROM THE PREVIOUS ESEM 2012 PAPER [11].

Table II (also from the ESEM 2012 paper) shows the correlation between increases and decreases in one metric outcome and the others as the system is refactored to optimise for a metric. As can be seen TCC, LSCC and SCOM exhibit collective moderate positive correlation, while CC and LCOM5 show mixed correlation ranging from moderate positive correlation (LCOM5 and SCOM) to strong negative correlation (LCOM5 and CC).

We further categorised each pair of metrics as follows:

Agreement:
Both metric values increase, both decrease, or both remains unchanged.

Dissonant:
One value increases or decreases, while the other remains unchanged.

Conflicted:
One value increases, while the other decreases.

Across the entire set of refactorings we studied, we found that 45% were in agreement, 17% were dissonant and a (surprisingly large) 38% were conflicted. The figure of 38% for the conflicted category is startling; how *can* all these metrics be measuring the same thing when they so often yield conflicting outcomes? These findings illustrate the value of our approach as a means of validating software metrics.

Our findings provide a technical underpinning for the oft-expressed concern that the attributes we seek to measure can be illusive and perhaps even partly unmeasurable. Such a high degree of conflict for metrics that are designed to measure the same thing, indicates that there is a problem.

Many a developer has reacted to a metrics programme with a degree of concern bordering on outright hostility. Perhaps such antagonism is justified. Our findings suggest that concern is well-founded, at least in the case of cohesion measurement.

More generally, our approach provides a means to establish the experimental scientific evidence to support (or refute) concerns about a set of related metrics. Much more research is needed to validate metrics and to identify those that can be stable.

We use cohesion merely as an illustration. The approach to experimental metric evaluation using 'SBSE and metametrics' can be applied to any set of software product metrics. SBSE is increasingly applied, not only to software products, such as code [19], [20], [21], designs [22], [23], [24], [25], [26] and test cases [27], [28] but to software process problems such as requirements analysis [29], [30] and project management [31], [32]. Therefore, an interesting avenue for future work lies in the use of SBSE on process simulations to experimentally compute novel metametrics that assess software process metrics.

	JHotDraw (1007)	JTar (115)	XOM (193)	JRDF (13)	JabRef (257)	JGraph (525)	ArtOfIllusion (593)	Gantt (750)	All (3453)
LSCC	96	99	100	92	99	100	99	96	98
TCC	86	53	97	46	61	72	84	71	78
SCOM	79	70	93	92	79	89	77	80	81
CC	100	98	100	92	99	100	100	99	100
LCOM5	100	100	100	100	100	100	100	99	100

TABLE I

METRIC VOLATILITY AS A PERCENTAGE. THIS SHOWS THE PERCENTAGE OF REFACTORINGS THAT CAUSED A CHANGE IN A METRIC. THE NUMBER IN PARENTHESES IS THE NUMBER OF REFACTORINGS THAT WERE PERFORMED ON THIS APPLICATION. RESULTS TAKEN FROM THE PREVIOUS ESEM 2012 PAPER [11].

Our initial application of this approach may have yielded findings that seem rather negative and dispiriting for the metrics community, but there is a positive message in our previous work. We believe that, armed with a means to evaluate metrics in a rigorous experimental context, we can move forward our understanding of what aspects of software can be measured and how best to measure them.

We also believe that metrics have a significant — as yet largely untapped — potential, *even* when they do *not* completely faithfully capture the attribute they seek to measure. This potential will be most keenly felt by the field of SBSE; the very field of work that we used to quantify the doubts over cohesion metrics. This is the potential of fast approximate metric surrogates, a topic to which we turn in the next section.

III. METRICS ARE FITNESS FUNCTIONS TOO II: THE NEED FOR DYNAMIC ADAPTIVE FAST APPROXIMATE METRICS

One of the foremost challenges for software metrics researchers is to construct suitable metrics that capture precisely, and as closely as possible, the attributes of software systems they seek to measure. All applications of software metrics hitherto investigated broadly envisage a scenario in which the metric guides the decision maker in their management of the software development process or their assessment of the processes and products involved in software development [13], [33].

As such, it makes sense for researchers to focus on accurately and precisely capturing the attributes of concern. Surely, it would be unthinkable to construct metrics that are deliberately more abstract (less precise) than they need to be; why would we want to define a metric that does not measure exactly what we want it to measure?

However, the research agenda of dynamic adaptive search based software engineering [12] creates a use-case for software metrics that has *exactly* this property: we need approximate metrics that can act as surrogates for more computationally expensive measurements. The surrogates we seek are metrics that retain some of the essence of the more computationally expensive metric, but which sacrifice some degree of precision for computational performance. The surrogate can thus be used to cheaply assess an approximate fitness to guide a search based approach for dynamic adaptivity.

Most SBSE fitness functions measure software properties on an ordinal scale [34]. It is rare to find examples of ratio or interval scale metrics in search based software engineering applications. Indeed, perhaps it is, generally speaking, rare to find such metrics in software engineering (search based or otherwise).

This prevalence of ordinal scale metrics means that we are afforded a considerable degree of flexibility in the search for approximate surrogate metrics. So long as we preserve ordering we shall be faithfully retaining the guidance of the more computationally expensive measurement.

Surrogate metrics for Speed: It has been repeatedly shown that many SBSE algorithms are highly robust: so long as the fitness function offers *some* guidance towards improvements in fitness, it can serve as the guidance for the search process [2]. A fitness function that is incorrect some of the time, but correct for the majority of invocations, retains some selection pressure towards fitter individuals; the search process will still tend to reach optima but will do so less quickly.

In traditional applications of computational search, there is a trade-off. We can use an approximate fitness function (that can be computed cheaply) and this may still guide us towards optima. We seek a suitable balance of the trade-off between the faithfulness of the fitness function surrogates and the alacrity with which they can be computed. With the right balance, the less faithful surrogate search process will reach sufficiently high quality results *faster* than the same search guided by the computationally expensive but completely faithful fitness function.

Though the surrogate fitness function might even mislead the search on some occasions, by promoting a less fit candidate solution over a truly fitter alternative, the overall effect is still positive due to the savings in fitness function computation time. This saving can be considerable because fitness function computation dominates the overall computational complexity in many cases. This observation about the value of fast surrogate fitness functions has motivated a great deal of work within the optimisation and computational search community targeting the definition of suitable fitness function surrogates (for example the work of Branke et al. [35]).

Software Engineering is Different: One might think that search based software engineering would be no different to any other optimisation problem. That is, we would need fitness function surrogates for precisely the same reasons. We would seek to investigate similar trade-offs for software engineering applications between faithfulness of fitness function surrogates and the overall alacrity of the search process.

However, software engineering is unlike any other engineering application of computational search. It has previously been argued that the virtual nature of software makes software engineering the ideal application for search based techniques [36]. This motivates the study of SBSE as a specific sub-discipline [37] at the interface of computational search and software engineering.

The argument goes like this: Computer software is unlike any other engineering material, due to its virtual nature (all other engineering materials being physical). Fitness functions can be computed *directly* on the engineering artefact to be optimised. By contrast, other engineering applications of optimisation require a *simulation* of a *model* of the engineering artefact. For these non-software-based engineering applications, fitness is computed, not on the engineering artefact itself, but on these simulations.

Adaptivity Needs New Metrics: In this paper we would like to extend this argument about the special 'search-friendly' nature of software further: it is not just the directness of fitness computation that makes search based software engineering so different from other potential engineering applications of computational search. The inherent potential for *adaptivity* of deployed software offers search based software engineering a unique potential: the engineering artefact can be dynamically optimised *as it is being used*. Few other engineering materials can be optimised in such a manner: in-situ once deployed.

Any adaptivity in a physical engineering materials is tightly constrained by the physical properties of the engineering material and there has to be a physical mechanism to realise the adaptation. Such adaptation will require considerable pre-planning and its execution may draw heavily on energy and other resources.

For example, some buildings adapt to the direction of sunlight, but this is a very limited and pre-determined set of changes (and may require mechanical work that consumes considerable energy). By contrast, software can potentially be adapted by dynamically re-configuring the code to meet new operating environments. This is the research agenda known as *dynamic adaptive search based software engineering* (DASBSE) [12].

Any approach that seeks dynamic adaptivity must necessarily compute many fitness evaluations between adaptations so surrogate fitness computation will need to be fast. The time between adaptations need not be instantaneous, but, depending on the context, adaptive optimisation computation may have to complete within days, hours, minutes or even seconds.

Example: Suppose we want to re-configure a system to adapt to fluctuations in power and bandwidth available on a smartphone. We can use multi-objective optimisation to construct a 'pareto program surface': the space of potential programs that trade off these different properties. This could be done at compile time to explore the design space with respect to non-functional properties [38].

However, what if we could compile that same optimisation capability into the deployed software system, so that the smartphone could dynamically adapt to changes in available power and bandwidth during operation?

We would need to run the optimisation as some form of background process. This process would need to evaluate potential modifications with regard to their likely bandwidth and power consumption. For either such property, it will not be practical to execute the modification; we shall merely be able to compute a fast approximate surrogate for these attributes.

This example is one of many in which non-functional properties are paramount and for which we might seek surrogate metrics. This agenda also creates interesting connections between software metrics, SBSE and predictive modelling [39].

IV. Appendix: Further Reading

For metrics researchers new to the field of SBSE and Dynamic Adaptive SBSE, this appendix provides some pointers to the literature.

There have been several surveys on of SBSE, each focussing on different aspects such as requirements [29], predictive modelling [40], SBSE for the cloud [41], machine learning and AI [42], design [26] and testing [43], [44], [45], [46]. Other recent surveys on mutation testing [47] and regression testing [48] contain sections describing the use of SBSE to attack the problems in these domains.

The relatively new area of Dynamic Adaptive SBSE is described in the ESEM 2012 keynote paper [12], while a more detailed description of the use of genetic programming to construct pareto program surfaces can be found in the ASE 2012 keynote paper [38].

There are also papers that set out open problems and future research agendas in SBSE for program comprehension [49], software maintenance [50], predictive modelling [39], testing [45], [51], bug fixing [52] and testability transformation [53].

A brief review of the growth in evolutionary computation for software engineering can be found in the IEEE Computer article [54]. For more comprehensive surveys of the whole field of SBSE, there is a complete survey [2], [55], which maps the entire SBSE area and provides trend analysis. While this paper presents a complete survey, there is also a ten year retrospective [37] that provides a shorter survey, focussed on a bibliometric analysis of the literature. There is also a detailed analysis and survey of the growth and development of SBSE in Brazil [56].

Finally, there are many tools that implement SBSE techniques, covering a range of applications including test data generation [27], [57], [58], [59], [60], modularisation [25], refactoring [61], bug fixing [62], mutation testing [63] and requirements optimisation [64].

V. The DAASE Project

The research agenda briefly outlined in this paper forms the focus of the DAASE project (DAASE: Dynamic Adaptive Automated Software Engineering, grant number EP/J017515).

DAASE is a major research initiative running from June 2012 to May 2018, funded by £6.8m from the Engineering and Physical Sciences Research Council (the EPSRC). DAASE also has matching support from University College London and the Universities of Birmingham, Stirling and York, which will complement the 22 EPSRC-funded post doctoral researchers recruited to DAASE with 26 fully funded PhD studentships and 6 permanent faculty positions (assistant and associate professors).

The DAASE project is keen to collaborate with leading researchers and research groups. We are also interested in collaboration with industrial parters and other organisations interested in joining the existing DAASE industrial partners which include AirFrance/KLM, Berner & Mattner, British Telecom, DSTL, Ericsson, GCHQ, Honda, IBM, Park Air Systems, Microsoft and Visa Europe. We have a programme for short and longer term visiting scholars (at all levels from PhD student to full professor) and arrangements for staff exchanges and internships with other organisations.

For more information, contact Lena Hierl, the DAASE Administrative Manager (`crest-admin@ucl.ac.uk`) or Mark Harman, the DAASE project director.

REFERENCES

[1] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, Dec. 2001.

[2] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering: Trends, techniques and applications," *ACM Computing Surveys*, vol. 45, no. 1, p. Article 11, November 2012.

[3] J. Clark, J. J. Dolado, M. Harman, R. M. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings — Software*, vol. 150, no. 3, pp. 161–175, 2003.

[4] 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*, B. Meyer and M. Nordio, Eds. Springer, 2012, pp. 1–59, LNCS 7007.

[5] M. Harman and J. Clark, "Metrics are fitness functions too," in 10^{th} *International Software Metrics Symposium (METRICS 2004)*. Los Alamitos, California, USA: IEEE Computer Society Press, Sep. 2004, pp. 58–69.

[6] S. Bouktif, G. Antoniol, E. Merlo, and M. Neteler, "A novel approach to optimize clone refactoring activity," in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, vol. 2. Seattle, Washington, USA: ACM Press, 8-12 Jul. 2006, pp. 1885–1892. [Online]. Available: http://www.cs.bham.ac.uk/~wbl/biblio/gecco2006/docs/p1885.pdf

[7] D. Fatiregun, M. Harman, and R. Hierons, "Search-based amorphous slicing," in 12^{th} *International Working Conference on Reverse Engineering (WCRE 05)*, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Nov. 2005, pp. 3–12.

[8] M. O'Keeffe and M. Ó Cinnédie, "Search-based refactoring: an empirical study," *Journal of Software Maintenance*, vol. 20, no. 5, pp. 345–364, 2008.

[9] M. Harman and L. Tratt, "Pareto optimal search-based refactoring at the design level," in *GECCO 2007: Proceedings of the* 9^{th} *annual conference on Genetic and evolutionary computation*. London, UK: ACM Press, Jul. 2007, pp. 1106 – 1113.

[10] O. Seng, J. Stammel, and D. Burkhart, "Search-based determination of refactorings for improving the class structure of object-oriented systems," in *Genetic and evolutionary computation conference (GECCO 2006)*, vol. 2. Seattle, Washington, USA: ACM Press, 8-12 Jul. 2006, pp. 1909–1916. [Online]. Available: http://www.cs.bham.ac.uk/~wbl/biblio/gecco2006/docs/p1909.pdf

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

[12] M. Harman, E. Burke, J. A. Clark, and X. Yao, "Dynamic adaptive search based software engineering," in 6^{th} *IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2012)*, Lund, Sweden, September 2012, pp. 1–8.

[13] M. J. Shepperd, *Foundations of software measurement*. Prentice Hall, 1995.

[14] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," in *Symposium on Software reusability (SSR '95)*, 1995, pp. 259–262.

[15] L. C. Briand, J. W. Daly, and J. Wüst, "A unified framework for cohesion measurement in object-oriented systems," *Empirical Software Engineering*, vol. 3, no. 1, pp. 65–117, 1998.

[16] C. Bonja, E. Kidanmariam, and E. Kidanmariam, "Metrics for class cohesion and similarity between methods," in *ACM Southeast Regional Conference*, 2006, pp. 91–95.

[17] L. Fernández and R. P. Na, "A sensitive metric of class cohesion," *Information Theories and Applications*, vol. 13, no. 1, pp. 82–91, 2006.

[18] J. Al-Dallal, L. C. Briand, and L. C. Briand, "An object-oriented high-level design-based class cohesion metric," *Information & Software Technology*, vol. 52, no. 12, pp. 1346–1361, 2010.

[19] A. Arcuri, D. R. White, J. A. Clark, and X. Yao, "Multi-objective improvement of software using co-evolution and smart seeding," in 7^{th} *International Conference on Simulated Evolution and Learning (SEAL 2008)*, ser. Lecture Notes in Computer Science, 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, Eds., vol. 5361. Melbourne, Australia: Springer, December 2008, pp. 61–70.

[20] W. B. Langdon and M. Harman, "Evolving a CUDA kernel from an nVidia template," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[21] W. Weimer, T. V. Nguyen, C. L. Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *International Conference on Software Engineering (ICSE 2009)*, Vancouver, Canada, 2009, pp. 364–374.

[22] C. L. Simons, I. C. Parmee, and R. Gwynllyw, "Interactive, evolutionary search in upstream object-oriented class design," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 798–816, 2010.

[23] K. Praditwong, M. Harman, and X. Yao, "Software module clustering as a multi-objective search problem," *IEEE Transactions on Software Engineering*, vol. 37, no. 2, pp. 264–282, 2011.

[24] M. Harman, S. Swift, and K. Mahdavi, "An empirical study of the robustness of two module clustering fitness functions," in *Genetic and Evolutionary Computation Conference (GECCO 2005)*. Washington DC, USA: Association for Computer Machinery, Jun. 2005, pp. 1029–1036.

[25] B. S. Mitchell and S. Mancoridis, "On the automatic modularization of software systems using the bunch tool," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 193–208, 2006.

[26] O. Räihä, "A survey on search–based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203–249, 2010.

[27] 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)*. ACM, September 5th - 9th 2011, pp. 416–419.

[28] 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*, vol. 55, no. 1, pp. 112–125, January 2013.

[29] 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 (REFSQ'08)*, vol. 5025. Montpellier, France: Springer LNCS, 2008, pp. 88–94.

[30] M. O. Saliu and G. Ruhe, "Bi-objective release planning for evolving software systems," in *Proceedings of the* 6^{th} *joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE) 2007*, I. Crnkovic and A. Bertolino, Eds. ACM, Sep. 2007, pp. 105–114.

[31] E. Alba and F. Chicano, "Software project management with GAs," *Information Sciences*, vol. 177, no. 11, pp. 2380–2401, June 2007.

[32] 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*, vol. 41, no. 5, pp. 495–519, April 2011.

[33] N. E. Fenton, *Software Metrics: A Rigorous Approach*. Chapman and Hall, 1990.

[34] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering 2007*, L. Briand and A. Wolf, Eds. Los Alamitos, California, USA: IEEE Computer Society Press, 2007, pp. 342–357.

[35] J. Branke and C. Schmidt, "Faster convergence by means of fitness estimation," *Soft Computing*, vol. 9, no. 1, pp. 13–20, 2005.

[36] M. Harman, "Why the virtual nature of software makes it ideal for search based optimization," in 13^{th} *International Conference on Fundamental Approaches to Software Engineering (FASE 2010)*, Paphos, Cyprus, March 2010, pp. 1–12.

[37] 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)*, 10th - 12th September 2011, pp. 18–32.

[38] 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)*, Essen, Germany, September 2012.

[39] M. Harman, "The relationship between search based software engineering and predictive modeling," in 6^{th} *International Conference on Predictive Models in Software Engineering (PROMISE 2010)*, Timisoara, Romania, 2010.

[40] W. Afzal and R. Torkar, "On the application of genetic programming for software engineering predictive modeling: A systematic review," *Expert Systems Applications*, vol. 38, no. 9, pp. 11 984–11 997, 2011.

[41] M. Harman, K. Lakhotia, J. Singer, D. White, and S. Yoo, "Cloud engineering is search based software engineering too," *Journal of Systems and Software*, 2012, to appear in print. Available online 23 November 2012.

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

[43] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, vol. 51, no. 6, pp. 957–976, 2009.

[44] 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*, pp. 742–762, 2010.

[45] M. Harman, "Automated test data generation using search based software engineering," in 2^{nd} *International Workshop on Automation of Software Test (AST 07)*. Minneapolis, USA: IEEE Computer Society Press, May 2007, p. 2.

[46] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–156, Jun. 2004.

[47] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649 – 678, September–October 2011.

[48] S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation: A survey," *Journal of Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[49] M. Harman, "Search based software engineering for program comprehension," in 15^{th} *International Conference on Program Comprehension (ICPC 07)*. Banff, Canada: IEEE Computer Society Press, 2007, pp. 3–13.

[50] M. D. Penta, "SBSE meets software maintenance: Achievements and open problems," in 4^{th} *International Symposium on Search Based Software Engineering (SSBSE 2012)*, ser. Lecture Notes in Computer Science, G. Fraser and J. T. de Souza, Eds., vol. 7515. Springer, 2012, pp. 27–28.

[51] P. McMinn, "Search-based software testing: Past, present and future," in *International Workshop on Search-Based Software Testing (SBST 2011)*. IEEE, 21 March 2011, pp. 153–163, keynote paper.

[52] C. L. Goues, S. Forrest, and W. Weimer, "Current challenges in automatic software repair," *Software Quality Journal*, 2013, to appear.

[53] M. Harman, "Open problems in testability transformation," in *1st International Workshop on Search Based Testing (SBT 2008)*, Lillehammer, Norway, 2008.

[54] ——, "Software engineering meets evolutionary computation," *IEEE Computer*, vol. 44, no. 10, pp. 31–39, Oct. 2011.

[55] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Department of Computer Science, King's College London, Tech. Rep. TR-09-03, April 2009.

[56] T. E. Colanzi, S. R. Vergilio, W. K. G. Assuno, and A. Pozo, "Search based software engineering: Review and analysis of the field in Brazil," *Journal of Systems and Software*, 2012, available online. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212002166

[57] 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)*, Lawrence, Kansas, USA, 6th - 10th November 2011, pp. 3 – 12.

[58] M. Harman, Y. Jia, and 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)*. New York, NY, USA: ACM, September 5th - 9th 2011, pp. 212–222. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025144

[59] K. Lakhotia, M. Harman, and H. Gross, "AUSTIN: A tool for search based software testing for the C language and its evaluation on deployed automotive systems," in 2^{nd} *International Symposium on Search Based Software Engineering (SSBSE 2010)*, Benevento, Italy, September 2010, pp. 101 – 110.

[60] P. Tonella, "Evolutionary testing of classes," in *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '04)*. Boston, Massachusetts, USA: ACM, 11-14 July 2004, pp. 119–128.

[61] 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)*, Honolulu, HI, USA, 2011, pp. 41–44.

[62] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A generic method for automatic software repair," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 54–72, 2012.

[63] 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)*, Windsor, UK, August 2008, pp. 94–98.

[64] 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*, vol. 12, no. 1, pp. 95–108, August 2008.