The Role of Artificial Intelligence in Software Engineering

Mark Harman
CREST Centre, University College London, Malet Place, London, WC1E 6BT, UK.

***Abstract*—There has been a recent surge in interest in the application of Artificial Intelligence (AI) techniques to Software Engineering (SE) problems. The work is typified by recent advances in Search Based Software Engineering, but also by long established work in Probabilistic reasoning and machine learning for Software Engineering. This paper explores some of the relationships between these strands of closely related work, arguing that they have much in common and sets out some future challenges in the area of AI for SE.**

I. Introduction

The history of the field of Artificial Intelligence (AI) is long and illustrious, tracing its roots back to the seminal work of Turing [1] and McCarthy [2]. The idea that machines can be intelligent has provided a staple diet for science fiction. Despite this, AI can also seem rather commonplace: Computational intelligence regularly provides examples of *specific areas* of intelligent behaviour for which machines comfortably surpass the performance of even the best human. Right from its intellectual origins in the 1950s the field stimulated philosophical as well as technological debate and raised much interest, not to mention a little concern, from the wider public.

Software engineers, by contrast, are less used to seeing their work in the science fiction literature. They are typically focussed on more prosaic and practical engineering concerns. Nevertheless, the software engineering research and practitioner communities have fallen under the 'AI spell'.

Artificial Intelligence is about making machines intelligent, while software engineering is the activity of defining, designing and deploying some of the most complex and challenging systems mankind has ever sought to engineer. Though software engineering is one of the most challenging of all engineering disciplines, it is often not recognised as such, because software is so well concealed.

Consider the Eiffel tower, a marvel of engineering reputedly containing no fewer than 2.5 million rivets [3]. It is an unmissable physical manifestation of engineering prowess, dominating the Paris skyline. By contrast, the scale of the engineering challenge posed by software remains entirely invisible. When one of the Eiffel Tower's 2.5 million rivets fails, the tower itself does not fail. Compare this enormous engineering edifice with a typical tiny smart phone, which may contain five to ten million lines of code, the failure of any one of which could lead to total system failure[1]. The space of inputs to even the smallest app on the phone is likely to comfortably exceed 10^{80} (a reasonable current estimate for the number of atoms in the observable universe), yet all but a single one of these inputs may fail to reveal the presence of just such a critical fault.

Faced with the daunting challenge of designing, building and testing engineering systems at these scales, software engineers fortunately have one critical advantage that other engineers do not posses; the software engineer's own material, software, can be used to attack the challenges posed by the production of systems in this very same material. AI algorithms are well suited to such complex software engineering problems, because they are designed to deal with one of the most demanding challenges of all; the replication of intelligent behaviour. Which software engineer would not want to have the assistance of intelligent software tools?

As a result of this natural technological pull, the software engineering community has adopted, adapted and exploited many of the practical algorithms, methods and techniques that have emerged from the AI community. These AI algorithms and techniques find important and effective applications that impact on almost every area of software engineering activity. In particular, the SE community has used three broad areas of AI techniques:

1) Computational search and optimisation techniques (the field known as Search Based Software Engineering (SBSE).
2) Fuzzy and probabilistic methods for reasoning in the presence of uncertainty.
3) Classification, learning and prediction.

Of course, neither Software Engineering nor Artificial Intelligence are static fields of activity; there is surely more to come. In the past five years there have been important breakthroughs in AI, with which previously insoluble challenges have been overcome [4]. The existing work has already amply demonstrated that there is considerable potential for Software Engineers to benefit from AI techniques. This paper provides a brief analysis of this development, highlighting general trends, shared and overlapping nomenclature, open problems and challenges.

It is perhaps tempting to categorise, compartmentalise and deconstruct the overall area of AI for SE into sub-domains. However, as we shall see there is considerable overlap between SE applications and applicable AI techniques and so this would be a mistake, albeit an appealing mistake for those whose professional life is spent studying classifiers!

This paper briefly reviews the three primary areas where AI techniques have been used in Software Engineering, showing their relationships and (considerable) overlap of aims and techniques. It concludes with five challenges that lie ahead in the development of AI for SE.

[1]It is important to recognise that there will be many lines of code that can be deleted with no observable effect of the behaviour of the device. However, we can be almost certain that there will exist a non-trivial set of statements, the deletion of any one of which would lead to a total system failure.

II. When does AI for SE Work Well?

The areas in which AI techniques have proved to be useful in software engineering research and practice can be characterised as 'Probabilistic Software Engineering', 'Classification, Learning and Prediction for Software Engineering' and 'Search Based Software Engineering'.

In Fuzzy and probabilistic work, the aim is to apply to Software Engineering, AI techniques developed to handle real world problems which are, by their nature, fuzzy and probabilistic. There is a natural fit here because, increasingly, software engineering needs to cater for fuzzy, ill-defined, noisy and incomplete information, as its applications reach further into our messy, fuzzy and ill-defined lives. This is not only true of the software systems we build, but the processes by which they are built, many of which are based on estimates.

One example of a probabilistic AI technique that has proved to be highly applicable in Software Engineering has been the use of Bayesian probabilistic reasoning to model software reliability [5], one of the earliest [6] examples of the adoption of what might be called, perhaps with hindsight, 'AI for SE'. Another example of the need for probabilistic reasoning comes from the analysis of users, inherently requiring an element of probability because of the stochastic nature of human behaviour [7].

In classification, learning and prediction work there has been great interest in modelling and predicting software costs as part of project planning. For example a wide variety of traditional machine learning techniques such as artificial neural networks, cased based reasoning and rule induction have been used for software project prediction [8], [9], ontology learning [10] and defect prediction [11]. An overview of machine learning techniques for software engineering can be found in the work of Menzies [12].

In Search Based Software Engineering (SBSE) work, the goal is to re-formulate software engineering problems as optimisation problems that can then be attacked with computational search [13], [14]. This has proved to be a widely applicable and successful approach, with applications from requirements and design [15], [16] to maintenance and testing [17], [18], [19]. Computational search has been exploited by all engineering disciplines, not just Software Engineering. However, the virtual character of software makes it an engineering material ideally suited to computational search [20]. There is a recent tutorial that provides a guide to SBSE [21].

III. Relationship between approaches to AI for SE

The various ways in which AI techniques have been applied in software engineering reveal considerable overlaps. For instance, the distinctions between probabilistic reasoning and prediction for software engineering is extremely blurred, if not rather arbitrary. One can easily think of a prediction system as nothing more than a probabilistic reasoner. One can also think of Bayesian models as learners and of classifiers as learners, probabilistic reasoners and/or optimisers.

Indeed, all of the ways in which AI has been applied to software engineering can be regarded as ways to optimise either the engineering process or its products and, as such, they are all examples of Search Based Software Engineering. That is, whether we think of our problem as one couched in probability, formulated as a prediction system or characterised by a need to learn from experience, we are always seeking to optimise the efficiency and effectiveness of our approach and to find good cost-benefit trade offs.

These optimisation goals can usually be formulated as measurable objectives and constraints, the solutions to which are likely to reside in large spaces, making them ripe for computational search.

There is very close interplay between machine learning approaches to Software Engineering and SBSE approaches. Machine learning is essentially the study of approaches to computation that improve with use. In order to improve, we need a way to measure improvement and, if we have this, then we can use SBSE to optimise according to it. Fortunately, in Software Engineering situations we typically have a large number of candidate measurements against which we might seek to improve [22].

Previous work on machine learning and SBSE also overlaps through the use of genetic programming as a technique to learn/optimise. Genetic programming has been one of the most widely used computational search techniques in SBSE work [23], with exciting recent breakthoughs in automatic bug fixing [24], [25] porting between platforms, languages and programming paradigms [26] and trading functional and non-functional properties [27].

However, genetic programming can also be thought of as an algorithm for learning models of software behaviour, a lens through which it appears to be a machine learning approach as well as an optimisation technique [28], [29]. Therefore, we can see that there are extremely close connections between machine learning for SE and SBSE: one way of learning is to optimise, while one way to think of the progress that takes place during optimisation is as a learning process.

Terminological arguments should not become a trap into which we fall, interminably and introspectively arguing over problem and solution demarkations. Rather, this rich shared and interwoven nomenclature can be regarded as an opportunity for exchange of ideas. For example SBSE can be used to optimise the performance of predictive models [30] and case based reasoners [31].

The first step for the successful application of any AI technique to any Software Engineering problem domain, is to find a suitable formulation of the software engineering problem so that AI techniques become applicable. Once this formulation is accomplished it typically opens a technological window of opportunity through which many AI techniques may profitably pass, as has been repeatedly demonstrated in previous work [18], [17], [19], [16], [15].

IV. Challenges Ahead in AI for SE

This section outlines some of the open problems in the application of AI techniques to Software Engineering.

A. Searching for strategies rather than instances

Current approaches to the application of AI to SE tend to focus on solving specific problem instances: the search for test data to cover a specific branch or a specific set of requirements or the fitting of an equation to predict the quality of a specific system. There is scope to move up the abstraction chain from problem instances to whole classes of problems and, from there, to the provision of strategies for finding solutions rather than the solutions themselves.

There has already been some initial work on ways of searching for derived probability distributions for statistical testing [32] and for inferring strategies from paths in model checking [33]. There has also been work on the search for tactics for program transformation [34], [35].

However, this work remains focussed on specific problems. It remains to be seen how we can best migrate from searching for solution instances to searching for strategies for finding solution instances. Through this avenue of future work, we shall be exploiting the natural connections between SBSE and machine learning, since the search for strategies can be thought of as a learning process over a training set.

Genetic Programming (GP), in particular, has the potential to generalise from the solution of problem instances to the solution of problem classes. Instead of searching for a test input to achieve a test goal, why not use genetic programming to characterise the strategies that find the next test input, based on the behaviour so-far observed. Rather than seeking a specific set of requirements for the next release of the software we might move closer to the original goals of strategic release planning [36]. That is, search for strategies to manage release of the software, charactersing the release strategy using GP.

B. Exploitation of Multicore Computation

A somewhat dated view of AI techniques might consider them to be highly computationally expensive, making them potentially unsuited to the large scale problems faced by software engineers. Fortunately, many of the AI techniques that we may seek to apply to Software Engineering problems, such as evolutionary algorithms, are classified as 'embarrassingly parallel'; they naturally decompose into sub-computations that can be carried out in parallel.

This possibility for parallelisation has been exploited in work on software re-modularisation [37], [38], concept location [39] and regression testing [40]. Although this work is very promising, more work is required to fully exploit the enormous potential of the rapidly increasing number of processors available.

One of the principal challenges for multicore computation remains the task of finding ways to translate existing programming paradigms into naturally parallisable versions [41]. This is essential if any speed up is to be achieved. Without it, execution on multicore can actually decrease performance, since each core is typically clocked at a lower rate than a similar single core system [42].

For many of the AI techniques discussed in this paper and almost all of those associated with SBSE, the algorithms used are naturally parallelisable. Yoo et al. [40] report that, with an inexpensive General Purpose Graphics Processing Unit (GPGPU), they are able to achieve speed ups over single computations of factors ranging up to 25. They also report that for larger regression testing problems the degree of scale up also tends to increase. The increasing number of processors available is an exciting prospect for scalability, chastened only by the observation our software engineering problems may be scaling at similar rates.

C. Giving Insight to Software Engineers

AI techniques do not merely provide another way to find solutions to software engineering problems, they also offer ways to yield insight into the nature of these problems and the spaces in which their solutions are to be found. For instance, though much work has been able to find good requirements [43], [44], project plans [45], [46], designs [47], [48] and test inputs [49], [50], [32], there is also much work that helps us to gain insight into the nature of these problems.

For instance, SBSE has been used to reveal the trade offs between requirements' stakeholders [51] and between requirements and their implementations [52] and to bring aesthetic judgements into the software design process [53]. There has also been work on understanding the risks involved in requirements miss-estimation and in project completion times [54], [55], while predictive models of faults, quality, performance and effort [56], [29], [57], [58] are naturally concerned with the provision of insight rather than solutions.

There remain many exciting and interesting ways in which AI techniques can be used to gain insight. For example some open problems concerning program comprehension are described elsewhere [59]. Such work is, of course, harder to evaluate than work which merely seeks to provide solutions to problems, since it involves measuring the effects of the AI techniques on the provision of insight, rather than against existing known best solutions. This is inherently more demanding, and the referees of such papers need to understand and allow for this elevated evaluation challenge. However, there is tremendous scope for progress; AI techniques have already been shown to outperform humans in several software engineering activities [60].

D. Compiling Smart Optimisation into Deployed Software

Most of the work on AI for SE, such as optimisation, prediction and learning has been applied off-line to improve either the software process (such as software production, designs and testing) or the software itself (automatically patching improving and porting). We might ask ourselves

> "If we can optimise a version of the system, why not compile the optimisation process into the deployed software so that it becomes dynamically adaptive?"

In order to deploy optimisation into software products, we need to identify the parameters that we should optimise [57], which could, itself be formulated as an optimisation problem. We might also speculate that work on genetic programming as a means of automatically patching, improving and porting software [24], [26], [25], [27], may be developed to provide *in situ* optimisation.

This would provide us with a set if tools and techniques with which to address long-standing challenges such as autonomic computing [61] and self-adapting systems [62].

E. Novel AI-Friendly Software Development and Deployment

We cannot expect to simply graft AI techniques into existing Software Engineering process and use-cases. We need to adapt the processes and products to better suit a software engineering world rich in the application of AI techniques. AI algorithms are already giving us intelligent software analysis, development, testing and decision support systems. These smart tools seek to support existing software development methods and processes, as constructed for largely human-intensive software development.

As the use of automated smart AI-inspired tools proliferates, we will need to rethink the best ways in which these can be incorporated into the software development process.

For instance, if faults can be automatically fixed, we need a release policy that accounts for this. Perhaps automated patches may not be, at least initially, so fully trusted as human-generated patches, then they might be used in tandem with the original system for ongoing regression testing.

If deployed software is able to take advantage of dynamic optimisation, *in situ*, then we may need to design software products that are able to seamlessly and unobtrusively monitor user 'comfort' and 'satisfaction' with the dynamically optimising code. Users will need a way to express their level of frustration and dissatisfaction with a system implicitly, simply by using it, without interfering with this use. It will be no use asking the user every few seconds whether they are happy; the system must be designed to continually monitor this, rather than merely monitoring the surrogate non-functional properties against which it seeks to optimise.

V. Conclusion

The rapid growth in interest in topics such as Search Based Software Engineering is a testimony to the appetite the Software Engineering community clearly has for AI techniques. This is not merely a capricious fashion. It is grounded in the way in which Software Engineering is, itself, becoming less of a craft and more of an engineering discipline.

For several decades we have been moving away from small, localised, insulated, bespoke, well-defined construction towards large-scale development and maintenance of connected, intelligent, complex, interactive systems. The engineering character of the problems we face as software engineers, such as noisy, partially- and ill- defined application domains with multiple competing, conflicting and changing objectives, is dragging us from an unrealistic utopia of perfect construction to the more realistic, but imperfect world of engineering optimisation.

This change in the nature of software forces us to change our development and deployment techniques. It should come as no surprise that AI techniques are proving to be well-suited to this changing world, since their inspiration comes from human intelligence; the archetype of a noisy, ill-defined, competing, conflicting, connected, complex, interactive system.

Acknowldegments: Mark Harman is funded by EPSRC grants GISMO (EP/I033688) RE-COST (EP/I010165) CREST Platform Grant (EP/G060525) DAASE Programme Grant (EP/J017515) and by the EU FITTEST project (257574) and by a Google Research Award.

Yuanyuan Zhang maintains a comprehensive repository on SBSE, with over 1,000 papers:

`http://crestweb.cs.ucl.ac.uk/ resources/sbse_repository/`

Bill Langdon maintains a similarly comprehensive repository on Genetic Programming, with over 7,000 papers:

`http://www.cs.bham.ac.uk/ wbl/biblio/`

These two repositories contain searchable lists of papers and authors and are both managed by a human, not an automated process, so the contents are likely to be carefully chosen to ensure high precision and recall of appropriate papers. Readers interested in predictive modeling will surely also find many useful materials in the PROMISE repository [63], while a more general overview survey of repositories can be found in the work of Rodriguez et al. [64].

References

[1] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 49, pp. 433–460, Jan. 01 1950.

[2] J. McCarthy, "Programs with common sense," in *Proceedings of the Symposium on Mechanisation of Thought Processes*, vol. 1. London: Her Majesty's Stationery Office, 1958, pp. 77–84.

[3] J. P. Cramer, *Almanac of Architecture and Design*. Atlanta: Greenway Communications, 2000.

[4] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer Go: Mnote Caril tree search and extensions," *Communications of the ACM*, vol. 55, no. 3, pp. 106–113, Mar. 2012.

[5] N. E. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski, and P. Krause, "On the effectiveness of early life cycle defect prediction with Bayesian Nets," *Empirical Software Engineering*, vol. 13, no. 5, pp. 499–537, 2008.

[6] B. Littlewood and J. L. Verrall, "A Bayesian reliability growth model for computer software," *Applied Statistics*, vol. 22, no. 3, pp. 332–346, 1973.

[7] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse, "The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. San Mateo: Morgan Kaufmann, Jul. 1998, pp. 256–265.

[8] A. Idri, T. M. Khoshgoftaar, and A. Abran, "Can neural networks be easily interpreted in software cost estimation?" Honolulu, Hawaii, p. 11621167, 2003.

[9] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, and S. Webster, "An investigation of machine learning based prediction systems," *The Journal of Systems and Software*, vol. 53, no. 1, pp. 23–29, Jul. 2000.

[10] A. Maedche and S. Staab, "Ontology learning for the semantic web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 72–79, 2001.

[11] V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, "Empirical assessment of machine learning based software defect prediction techniques," *International Journal on Artificial Intelligence Tools*, vol. 17, no. 2, pp. 389–400, 2008.

[12] T. Menzies, "Practical machine learning for software engineering and knowledge engineering," in *Handbook of Software Engineering and Knowledge Engineering*. World-Scientific, December 2001, available from http://menzies.us/pdf/00ml.pdf.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering trends, techniques and applications," *ACM Computing Surveys*, 2012, to appear.

[20] 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.

[21] 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.

[22] 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.

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

[24] A. Arcuri and X. Yao, "A Novel Co-evolutionary Approach to Automatic Software Bug Fixing," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '08)*. Hongkong, China: IEEE Computer Society, 1-6 June 2008, pp. 162–168.

[25] 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.

[26] 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.

[27] D. R. White, J. Clark, J. Jacob, and S. Poulding, "Searching for resource-efficient programs: Low-power pseudorandom number generators," in *2008 Genetic and Evolutionary Computation Conference (GECCO 2008)*. Atlanta, USA: ACM Press, Jul. 2008, pp. 1775–1782.

[28] E. O. Costa, A. Pozo, and S. R. Vergilio, "A genetic programming approach for software reliability modeling," *IEEE Transactions on Reliability*, vol. 59, no. 1, pp. 222–230, 2010.

[29] J. J. Dolado, "On the problem of the software cost function," *Information and Software Technology*, vol. 43, no. 1, pp. 61–72, Jan. 2001.

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

[31] C. Kirsopp, M. Shepperd, and J. Hart, "Search heuristics, case-based reasoning and software project effort prediction," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. San Francisco, CA 94104, USA: Morgan Kaufmann Publishers, 9-13 July 2002, pp. 1367–1374.

[32] S. M. Poulding and J. A. Clark, "Efficient software verification: Statistical testing using automated search," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 763–777, 2010.

[33] J. Staunton and J. A. Clark, "Finding short counterexamples in Promela models using estimation of distribution algorithms," in 13^{th} *Annual Genetic and Evolutionary Computation Conference (GECCO 2011)*, N. Krasnogor and P. L. Lanzi, Eds. Dublin, Ireland: ACM, July 12th-16th 2011, pp. 1923–1930.

[34] D. Fatiregun, M. Harman, and R. Hierons, "Evolving transformation sequences using genetic algorithms," in 4^{th} *International Workshop on Source Code Analysis and Manipulation (SCAM 04)*. Los Alamitos, California, USA: IEEE Computer Society Press, Sep. 2004, pp. 65–74.

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

[36] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information & Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.

[37] B. S. Mitchell, M. Traverso, and S. Mancoridis, "An architecture for distributing the computation of software clustering algorithms," in *IEEE/IFIP Proceedings of the Working Conference on Software Architecture (WICSA '01)*. Amsterdam, Netherlands: IEEE Computer Society, 2001, pp. 181–190.

[38] K. Mahdavi, M. Harman, and R. M. Hierons, "A multiple hill climbing approach to software module clustering," in *IEEE International Conference on Software Maintenance*. Los Alamitos, California, USA: IEEE Computer Society Press, Sep. 2003, pp. 315–324.

[39] F. Asadi, G. Antoniol, and Y. Guéhéneuc, "Concept location with genetic algorithms: A comparison of four distributed architectures," in *Proceedings of* 2^{nd} *International Symposium on Search based Software Engineering (SSBSE 2010)*. Benevento, Italy: IEEE Computer Society Press, 2010, pp. 153–162.

[40] S. Yoo, M. Harman, and S. Ur, "Highly scalable multi-objective test suite minimisation using graphics cards," in 3^{rd} *International Symposium on Search based Software Engineering (SSBSE 2011)*, 10th - 12th September 2011, pp. 219–236, LNCS Volume 6956.

[41] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: a programming model for heterogeneous multi-core systems," in 13^{th} *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Seattle, WA, USA: ACM, Mar. 2008, pp. 287–296.

[42] A. Buttari, J. Dongarra, J. Kurzak, J. Langou, P. Luszczek, and S. Tomov, "The impact of multicore on math software," in 8^{th} *International Workshop Applied Parallel Computing (PARA 2006)*, vol. LNCS 4699. Umea, Sweden: Springer, June 2006, pp. 1–10.

[43] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, no. 5, pp. 67–74, September/October 1997.

[44] Y. Zhang, M. Harman, A. Finkelstein, and A. Mansouri, "Comparing the performance of metaheuristics for the analysis of multi-stakeholder tradeoffs in requirements optimisation," *Journal of Information and Software Technology*, vol. 53, no. 7, pp. 761–773, 2011.

[45] A. Barreto, M. Barros, and C. Werner, "Staffing a software project: A constraint satisfaction and optimization based approach," *Computers and Operations Research (COR) focused issue on Search Based Software Engineeering*, vol. 35, no. 10, p. 30733089, October 2008.

[46] 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.

[47] S. Mancoridis, B. S. Mitchell, Y.-F. Chen, and E. R. Gansner, "Bunch: A clustering tool for the recovery and maintenance of software system structures," in *Proceedings; IEEE International Conference on Software Maintenance*. IEEE Computer Society Press, 1999, pp. 50–59.

[48] 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.

[49] G. Fraser and A. Arcuri, "Evolutionary generation of whole test suites," in 11^{th} *International Conference on Quality Software (QSIC)*, M. Núñez, R. M. Hierons, and M. G. Merayo, Eds. Madrid, Spain: IEEE Computer Society, July 2011, pp. 31–40.

[50] M. Harman and P. McMinn, "A theoretical and empirical study of search based testing: Local, global and hybrid search," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 226–247, 2010.

[51] A. Finkelstein, M. Harman, A. Mansouri, J. Ren, and Y. Zhang, "A search based approach to fairness analysis in requirements assignments to aid negotiation, mediation and decision making," *Requirements Engineering*, vol. 14, no. 4, pp. 231–245, 2009.

[52] 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.

[53] 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.

[54] 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)*, Montreal, Canada, 8th – 12th July 2009, pp. 1673–1680.

[55] M. Harman, J. Krinke, J. Ren, and S. Yoo, "Search based data sensitivity analysis applied to requirement engineering," in *ACM Genetic and Evolutionary Computation COnference (GECCO 2009)*, Montreal, Canada, 8th – 12th July 2009, pp. 1681–1688.

[56] S. Bouktif, H. Sahraoui, and G. Antoniol, "Simulated annealing for improving software quality prediction," 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. 1893–1900.

[57] K. Krogmann, M. Kuperberg, and R. Reussner, "Using genetic search for reverse engineering of parametric behaviour models for performance prediction," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 865–877, November-December 2010.

[58] D. Rodriguez, R. Ruiz, J. C. Riquelme-Santos, and R. Harrison, "Subgroup discovery for defect prediction," in 3^{rd} *International Symposium on Search Based Software Engineering (SSBSE)*, vol. 6956. Szeged, Hungary: Springer, September 2011, pp. 269–270.

[59] 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.

[60] J. Souza, C. L. Maia, F. G. de Freitas, and D. P. Coutinho, "The human competitiveness of search based software engineering," in *Proceedings of* 2^{nd} *International Symposium on Search based Software Engineering (SSBSE 2010)*. Benevento, Italy: IEEE Computer Society Press, 2010, pp. 143 – 152.

[61] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[62] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, vol. 24, no. 2, pp. 163–186, 2012.

[63] G. Boetticher, T. Menzies, and T. Ostrand, "PROMISE repository of empirical software engineering data," 2007, available at http://promisedata.org/ repository.

[64] D. Rodriguez, I. Herraiz, and R. Harrison, "On software engineering repositories and their open problems," in *The International Workshop on Realizing AI Synergies in Software Engineering (RAISE'12)*, Zurich, Switzerland, 2012.