# Search Methodologies in Real-world Software Engineering

Gabriela Ochoa
Computing Science and Mathematics
University of Stirling
Stirling, Scotland,UK
gabriela.ochoa@cs.stir.ac.uk

## ABSTRACT

One of the aims of software engineering is to reduce overall software costs. Optimisation is, therefore, relevant to the process of software development. This article describes recent case studies on the application of modern search methodologies to challenging real-world problems in software engineering. It also describes a recent research initiative: *Dynamic Adaptive Automated Software Engineering (DAASE)*, whose goal is to embed optimisation into deployed software to create self-optimising adaptive systems. The article accompanies an invited talk for the Workshop on *Bridging the Gap between Industry and Academia in Optimisation* to be held as part of GECCO 2013.

## Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

## General Terms

Algorithms, Design, Performance, Experimentation, Verification

## Keywords

search based software engineering; evolutionary algorithms; hyper-heuristics; metaheuristics; adaptive systems

## 1. INTRODUCTION

Software engineering is a discipline within computer science aiming to produce quality software, delivered on time, within budget and that covers the users's needs. This is a difficult task as software systems are large and complex, built by teams, exist in many versions, last several years and undergo changes. The process of software development is commonly called the *software life-cycle* and it involves the following phases, according to the well-known *water-fall* model: (i) requirements analysis and specification, (ii) design and specification, (ii) code and module testing, (iii) integration and system testing, and (iv) delivery and maintenance. It is useful to consider these stages as distinct for management purposes, but in practice they overlap and feed information to each other.

Software engineering is not just about producing software systems, but aims at producing them in the most effective way. The challenge is to produce high-quality software with a finite amount of resources. There is, therefore, a wide scope for applying optimisation techniques in all stages of the software life-cycle. Consider, for example, the following illustrative questions from [9]: "(1) What is the smallest set of test cases that covers all the branches in this program?, (2) What is the best way to structure the architecture of this system to enhance its maintainability?, (3) What is the set of requirements that balances software development cost and customer satisfaction?". These an other problems in software engineering can be formulated as optimisation problems and solutions are found using modern search methodologies such as evolutionary algorithms and other metaheuristics.

The term *Search-based Software Engineering (SBSE)* was coined in 2001 by Harman and Jones [10]. Since then there has been an explosion of activity in this area with an increasing number of publications appearing in prestigious journals and conferences [9].

This short article describes 3 case studies applying search methodologies to challenging problems in software engineering. It also describes a recent research initiative: *Dynamic Adaptive Automated Software Engineering (DAASE)*, whose goal is to embed optimisation into deployed software to create self-optimising adaptive systems [7].

## 2. CASE STUDIES

### 2.1 Requirements Optimisation

The requirements of a software system are simply a detailed statement of the things that the system must be able to do. The users' needs or conditions should be clearly specified for either new or altered software systems. When doing so, the possibility of conflicting requirements of the various stakeholders should be considered. One important goal is to select near-optima subsets of all possible requirements to satisfy the customer demands, while securing that sufficient resources are available to undertake the selected tasks [19]. Requirements decisions can be formulated as optimisation problems. For example, in the development and maintenance of large, complex software systems sold to a range of diverse customers, a problem faced is that of determining what should be in the next release of the software. In the *Next Release Problem*, the goal is to find the ideal set of requirements that balance customer requests within resource constraints. This problem is formulated by Bagnall et al. [2] as a constrained single objective optimisation problem and successfully solved using metaheuristics. A multi-objective formulation is presented by Zhang et al. [20] where evolutionary methods are used to find approximations of the Pareto-optimal set. This allows the decision maker to select the preferred solution from this set according to their priorities. The Pareto set also provides valuable insights into the outcome of the selected set of requirements to the software development company, as it captures the trade-offs between competing objectives.

## 2.2 Search-Based Test Data Generation

Software testing is a critical element of software quality assurance. Once source code has been generated, software must be tested to uncover as many errors as possible. This is accomplished by designing a series of test cases that have a high likelihood of finding errors. Designing a good test set is challenging, and many techniques and strategies have been proposed. Search-based approaches have been applied to several testing goals and issues [9]. An interesting topic is the automated generation of test cases through search [13]. An example can be found within the so called *statistical testing*, where test data is generated by sampling from a probability distribution defined over the software's input domain. The distribution needs to be carefully chosen so that it satisfies the testing objectives expressed in terms of functional or structural properties of the software. Traditionally, the distributions are manually chosen. In order to automate this process, Poulding et al. [17] use search in the space of probability distributions (which are represented as Bayesian Networks) for statistical testing. The results show that the approach is viable and practical, producing superior fault-detection ability than other forms of testing in the cases studied. The method is, however, limited to a fixed number of numeric values as the software inputs. To overcome this limitation, a more flexible representation of probability distributions based on stochastic grammars is presented in [16]. With this new representation the algorithm can be applied to a wider range of software, in particular software with structurally-complex inputs. For three real-world examples studied, the algorithm took only a few minutes to derive suitable input profiles using computing resources equivalent of a desktop PC.

## 2.3 Automatic Program Repair and Improvement

An exciting recent application of search methodologies is that of automatically repairing and improving programs directly using the source code, a test suite, and without relying on formal specifications. The idea of using genetic programming to repair software bugs was initially proposed in [1]. More substantial experimental results on real programs and real bugs have been carried out in [12], where off-the-shelf programs are repaired with a combination of program analysis methods with evolutionary computation.

Search techniques have also been used to automatically improve the behaviour of a software system with respect to some desired criteria, usually related to non-functional properties such as execution time, program size, throughput, power consumption and bandwidth [8, 11, 15, 18]. The functional properties of the system should be maintained as faithfully as possible. This is achieved by evaluating the modified programs with a fitness function based on a set of test cases from the original system. Thus, the original system acts as an oracle of the modified, improved versions. This automated approach has many potential applications, such as porting existing programs from one platform to another and producing programs with faster execution time or less power consumption, while still performing the core required functionality of the original system.

## 3. THE *DAASE* PROJECT

The case studies mentioned in this short article are a small sample of the research focus of the DAASE (Dynamic Adaptive Automated Software Engineering) project [7]. The project aims to enhance current successful applications of search and optimisation in software engineering by strengthening aspects of adaptivity, automation, robustness and coping with dynamic environments. Instrumental to the project is the application of hyper-heuristics [4, 5, 14] and other forms of autonomous search [3, 6]. 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* (EPSRC, grant number EP/J017515), with matching support from 4 institutions in the UK: University College London, and the Universities of Birmingham, Stirling and York. It also has a growing number of industrial partners including AirFrance/KLM, Berner & Mattner, British Telecom, DSTL, Ericsson, GCHQ, Honda, IBM, Park Air Systems, Microsoft and Visa Europe. The DAASE project fosters collaborations with leading researchers and research groups, having a program to support both visiting scholars at all levels and staff interchanges with other organisations. For further information, please visit the project website `http://daase.cs.ucl.ac.uk/`.

## 4. REFERENCES

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

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

[3] R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations Research/Computer Science Interfaces Series*. Springer, 2009.

[4] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.

[5] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer, 2010.

[6] Y. Hamadi, E. Monfroy, and F. Saubion, editors. *Autonomous Search*. Springer, 2012.

[7] M. Harman, E. Burke, J. A. Clark, and X. Yao. Dynamic adaptive search based software engineering. In *Proceedings of the 6th International Symposium on Empirical Software Engineering and Measurement (ESEM '12) (Keynote)*, Lund, Sweden, 19-20 September 2012. ACM.

[8] M. Harman, W. Langdon, Y. Jia, D. White, A. Arcuri, and J. Clark. The gismoe challenge: Constructing the pareto program surface using genetic programming to find better programs. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE '12) (Keynote)*, Essen, Germany, 3-7 September 2012. ACM.

[9] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1), November 2012.

[10] M. Harman, U. Ph, and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43:833–839, 2001.

[11] W. B. Langdon and M. Harman. Evolving a cuda kernel from an nvidia template. In *Proceedings of the IEEE World Congress on Computational Intelligence, CEC 2010*, pages 2376–Ű2383. IEEE Press, 2010.

[12] 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.

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

[14] G. Ochoa, J. Walker, M. Hyde, and T. Curtois. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 418–427. Springer, 2012.

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

[16] S. Poulding, R. Alexander, J. Clark, and M. Hadley. The optimisation of stochastic grammars to enable cost-effective probabilistic structural testing. In *Proceedings of the International Conference on Genetic and evolutionary computation conference, GECCO 2013*, page (to appear). ACM, 2013.

[17] S. Poulding and J. Clark. Efficient software verification: Statistical testing using automated search. *IEEE Transactions on Software Engineering*, 36(6):763–777, 2010.

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

[19] Y. Zhang, A. Finkelstein, and M. Harman. Search based requirements optimisation: Existing work and challenges. In *Proceedings of the 14th International Working Conference, Requirements Engineering: Foundation for Software Quality (RefsQ '08)*, volume 5025, pages 88–94, Montpellier, France, 16-17 June 2008. Springer.

[20] Y. Zhang, M. Harman, and S. Mansouri. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1129–1137, New York, NY, USA, 2007. ACM.