Software Engineering Experts' Panel: The Status and Future of SBSE in the Software Engineering Community

May 14, 2009, Cumberland Lodge, Windsor, UK

We are delighted to be hosting a panel discussion featuring five of the most prominent software engineering scientists in the world: Norman Fenton, Anthony Finkelstein, Paola Inverardi, Mary Lou Soffa, and Ian Sommerville.

The panel will provide their points of view on issues that include:

- How can search-based optimization techniques assist with problems that software engineers have struggled with for years?
- What are the main problems and limitations in applying search-based techniques to software engineering problems?
- What will be the challenges in applying search-based techniques to emerging areas of software engineering research and development?

We are sure that the discussion led by this distinguished panel will help to inform the direction that the SBSE community takes as it continues to grow. We encourage all attendees to take an active part in the discussion.

Mark Harman, *General Chair*
Massimiliano Di Penta and Simon Poulding, *Program Co-chairs*

Norman Fenton, Professor of Computer Science, Queen Mary, University of London, UK. During the 1970's and 1980s there were many computer scientists - and software engineers especially - who dreamed of perfect solutions to complex problems based purely on mathematical semantics and logic. With the right formal notation and proof rules they imagined that it would be possible, for example, to create complex software systems automatically from mathematical specifications that were perfect in the sense of requiring no testing [1]. Although for many this dream has never died, and examples of modestly complex, verified systems and tools to support them have been developed, in practice most software engineering challenges can never support perfect solutions [2]. Since even a perfectly verified system can only have been verified against a formal specification, the need for testing still remains for such a system since there can never be a formal automated means of verifying real-world requirements.
The quest for perfect solutions is, of course, also provably infeasible for a wide class of algorithmic problems, namely all of those such as the traveling salesman problem with complexity NP-complete or worse. Since such problems are pervasive and will not just go away, we have been forced to consider solutions that were less than perfect. This led to the broad discipline of approximation in algorithm design with different approaches to optimization and also probabilistic solutions [3]. It seems to me that the current discipline of search-based software engineering [4] is the inevitable extension of this work into areas beyond algorithm design, with a special emphasis on testing. The growing popularity of search-based software engineering therefore seems to be a sign of the maturity of the entire software engineering discipline as it finally moves away from the naivety of those earlier years and confronts the reality of accepting uncertainty and approximation.
While the need for search-based techniques to support testing of complex systems seems to me to be self-evident, my view is that the subject as a whole may be cast in too narrow a light, with too much focus on specific types of metaheuristic search [5]. I can understand why, to date the emphasis has been so. The need for optimising test data is both an obviously important problem and a natural candidate for the class of metaheuristic search techniques commonly used. However, as is made clear in [4] the challenges of search-based software engineering are far greater and could encompass a much broader range of methods. Search-based software engineering essentially deals with any software engineering problem for which there is no perfect solution and for which no efficient deterministic algorithm will find an optimal solution. Such problems are inevitably characterized by uncertainty, and I feel there is a much broader class of methods, which can be classified as 'intelligent', that may be relevant. At this point I will declare a blatant interest in one such method - Bayesian networks

(BNs)– that has been increasingly used to address decision problems involving uncertainty in software engineering [6,7,8,9]. In my background reading for this position statement I was actually surprised to discover that BNs comfortably satisfied the two key ingredients of search-based optimization for a range of software engineering problems, namely:

1. Choice of the representation of the problem
2. Definition of the fitness function

Moreover, BN solutions are also addressing some of the key open problems and challenges in optimization; for example, recent developments on dynamic discretisation algorithms for BNs partially solve the general challenge of stopping criteria cited in [4], while BNs are particularly powerful for addressing multi-objective optimization and sensitivity analysis. The latter are subjects which [4] cites as being part of the road map for future work.

Given that BNs can so easily fit into the criteria (and rationale) for search-based software engineering (while never having formally been considered a candidate), I suspect that there are many other methods beyond the narrow class of metaheuristic search which fit just as well. The big question is: do the current community of search-based software engineering researchers want to move beyond their current boundaries? There are always both benefits and disadvantages of a narrow focus.

Anthony Finkelstein, Professor of Software Systems Engineering, University College London, UK. What is the prospect for search based software engineering within the broader software engineering community? To address this question satisfactorily requires us to address the broader question of the medium and long-term directions for software engineering research. The challenges appear to be manifold. First disciplinary, how are we to address the potential fragmentation of the discipline into sub-specialisations (requirements, architectures, testing, analysis ...) and similarly into domains (enterprise systems, automotive systems, biomedical systems ...). And, how are we to address a growing divide between practitioners and researchers. Second technological, the limits of Moore's law are evident and we are faced with new technical challenges associated with massive fine grain parallelism and virtualization. We are increasingly faced with problems of scale (data, users, distribution ...) hitherto unimagined. We have always designed without regard to energy and associated resources, this must also be rethought. Thirdly conceptual, there are basic issues of disciplinary scope that have not been resolved, or rather have been set aside, in software engineering and which require revisiting.

In my panel contribution I intend to address some of these 'big' issues in software engineering and will speculate on the agenda for SBSE in this context. I hope in particular to suggest some new areas with great promise that might attract the attention of researchers with relevant 'toolkits'.

Paola Inverardi, Professor of Computer Science, Università dell'Aquila, Italy. *"Computing in the Ubiquitous world or Exploring the unknown"* **.** The design and the development of dependable and adaptable software applications in the near ubiquitous future (Softure) cannot rely on the classical desktop-centric assumption that the system execution environment is known a priori at design time and, hence, the application environment of a Softure cannot be statically anticipated. Softure will need to cope with variability, as software systems get deployed on an increasingly large diversity of computing platforms and operates in different execution environments. Heterogeneity of the underlying communication and computing infrastructure, mobility inducing changes to the execution environments and therefore changes to the availability of resources and continuously evolving requirements require software systems to be adaptable according to the context changes. At the same time, Softure should be reliable and meet the users performance requirements and needs. Moreover, due to its pervasiveness and in order to make adaptation effective and successful, adaptation must be considered in conjunction with dependability, i.e., no matter what adaptation is

performed, the system must continue to guarantee a certain degree of Quality of Service (QoS). Hence, Softure must also be dependable, which is made more complex given the highly dynamic nature of service provision.
Supporting the development and execution of Softure systems raises numerous challenges that involve languages, methods and tools for the systems through design and validation in order to ensure dependability of the self-adaptive systems that are targeted. To face these challenges Softure requires to rethink the whole software engineering process and, in particular, it needs to reconcile the static view with the dynamic view by breaking the traditional division among development phases by moving some activities from design time to deployment and run time hence asking for new and more efficient verification and validation techniques.
In this general setting there is plenty of room for exploring the use of SBSE techniques. Softure systems should be able to declaratively express their potential variability to the execution contexts allowing for adaptations/configurations that are a best match with respect to the (available resources in the) actual execution context and to the system QoS requirements. This variability is not only expressed at the system code level but, due to the dependability requirements, should also be reflected at the systems model level that must correspondingly adapt. These gives us at least two different but related search spaces (the code/program space and the model space) that must be dynamically explored to correctly drive the adaptation. In each search space the fitness function is indeed different. For example in the program space, given a certain resource context, we look for the program instances that can best/better exploit the available resources in terms of user satisfaction, while in the model space the same exploration looks for a system model that reflects the adaptation but still guarantees efficient verification. Indeed, taking a more comprehensive software engineering perspective we might need to consider adaptation and therefore variability applied at different level of abstractions including design artefacts and software architectures. SBSE techniques appear to be natural candidates to help in solving these problems. For SBSE the challenges here are therefore to be able to consistently co-relate multiple searches and, more importantly, to make the searches efficient in a highly dynamic context.

Mary Lou Soffa, Owen R. Cheatham Professor, University of Virginia, USA. Search Based Software Engineering (SBSE) emerged as an independent research and practice area about ten years ago. Since then, SB techniques have been applied to a variety of software engineering tasks, including testing, maintenance, program comprehension, and automatic programming, and cost estimation. There have also been more SB techniques included to tackle the hard problems in software engineering. However, challenges remain that I believe have to be addressed before there is wide spread – any commercial – use of SBSE.
First of all, experimentation is very necessary but is very time consuming when you need to evaluate different techniques (Random, HC, GA, SANN, ...), method configurations, and case study applications over many trials. There are no (or few) standard tools that help a user design and conduct experiments with SB techniques. It is also very difficult, and again necessary, to fairly compare results from different SB techniques and SB techniques with non-SB techniques.
The SB techniques themselves are complex and difficult to design. As soon as you start to use genetic algorithms or other sophisticated search techniques, you have to consider many different operators and parameters. It is often challenging to predict how different combinations of parameters will impact the efficiency and effectiveness of the technique.
Given the substantial number of configurations, etc., you often end up with large and multi-variate data sets that are difficult to analyze with traditional methods. In many cases, data mining techniques (e.g., classification and regression trees) are needed to identify trends in results. Because of these challenges, the "state of the art" in research publishing often leads to papers that do not publish efficiency results for search-based methods!
Because search-based methods are normally very expensive to run, you have to be careful about when you decide to use this type of approach. Sometimes experiments have shown that for problems with even moderate sized search spaces there was no clear reason to use search-based methods instead of greedy algorithms. I believe that we need to carefully think about the characteristics of a problem that lends itself to search based techniques, and only use it for these types of problems – that is, we should be careful not to oversell the idea.

Ian Sommerville, Professor of Software Engineering, University of St Andrews, UK. *"Search-based Software Engineering: Putting it into Practice".* The current situation of SBSE reminds me of the situation of the Formal Methods (FM) research community in the late 1970s. Then, we had a new software technology that showed promise with strong academic support and some industrial enthusiasts. However, in my view, the FM community as a whole then made strategic errors that meant that it took more than 20 years before FM any kind of impact and, even now, its impact on practical software development is and will probably remain limited.

So, where did the FM community go wrong. Leaving aside the issue of hubris which I don't see in the SBSE community, they focused on the problems that they could address rather than the real problems facing large system developers. They were concerned with correctness with respect to a specification, which was only an issue for a tiny fraction of systems and did not investigate how to use their technology to understand requirements, an issue for all systems. They largely ignored the risks of using formal methods in practical projects and did not address the problem of using FM in conjunction with existing technologies. They did not scope their approach and failed to address the problems of scale and providing long-term support for the technology.

My overall impression is that SBSE has already started to make some of the same mistakes. It uses a grand title (search based software engineering), but only addresses a small number of software engineering sub-problems; there is little discussion of how to use the technique in a practical process, the risks or the value that ensues. If you care about the practical use of SBSE, your challenge is to learn from the experience and to address the issues of utilisation of the approach before the technology reaches the 'trough of disillusionment'. The key requirements are to understand where the technology may be practically useful and to explore in depth how it can be applied in these areas and to understand the real costs and benefits which may result. It is essential to understand the human, social and organisational problems of technology transfer and to bear these in mind when designing new methods. Understand the risks of introducing and using new technologies and be able to explain how these risks can be mitigated.

References

[1] Hoare, C. A. R. (1986). Maths adds safety to computer programs. New Scientist, pp 53--56, Sept.

[2] Glass, R. L. (2002). "The proof of correctness wars." Commun. ACM **45**(8): 19-21.

[3] Harel, D. (1992). Algorithmics, Addison Wesley

[4] Harman, M., *The Current State and Future of Search Based Software Engineering*, in *2007 Future of Software Engineering*. 2007, IEEE Computer Society. p. 342-357.

[5] Harman, M. and Jones, B. (2001). SEMINAL: Software Engineering using Metaheuristic INnovative ALgorithms. 23rd IEEE/ACM International Conference on Software Engineering (ICSE 2001). Toronto, Canada, May 12th-13th, 2001**:** 762-763.

[6] de Melo, A. C. V. and Sanchez, A. J. (2008). "Software maintenance project delays prediction using Bayesian Networks." Expert Syst. Appl. **34**(2): 908-919, from http://dx.doi.org/10.1016/j.eswa.2006.10.040.

[7] Fenton, N. E., Marsh, W., Neil, M., Cates, P., Forey, S. and Tailor, M. (2004). Making Resource Decisions for Software Projects. 26th International Conference on Software Engineering (ICSE2004) Edinburgh, United Kingdom, IEEE Computer Society**:** 397-406.

[8] Fenton, N. E., et al. (2008). "On the effectiveness of early life cycle defect prediction with Bayesian Nets." Empirical Software Engineering **13**: 499-537.

[9] Stamelosa, I., Angelisa, L., Dimoua, P. and Sakellaris, P. (2005). "On the use of Bayesian belief networks for the prediction of software productivity." Information and Software Tech **45** (1): 51-60.

[10] Neil, M., Tailor, M. and Marquez, D. (2007). "Inference in hybrid Bayesian networks using dynamic discretization." Statistics and Computing **17**(3): 219-233, from http://dx.doi.org/10.1007/s11222-007-9018-y.