Abstract—This challenges and open problems paper discusses the role of software engineers when combining modeling with Search-Based Software Engineering (SBSE). With SBSE techniques increasingly providing insights into difficult software modeling problems, the role of the software engineer might be thought of as diminishing. However, given the crucial role of the model understanding in SBSE, we argue that the role of the software engineer will become more important through interactive SBSE, although exploiting this interaction remains a challenge for the field.

Index Terms—Software Engineers, Search-Based Software Engineering, Interaction.

Since its emergence as an optimization technique for difficult software engineering problems, Search-Based Software Engineering (SBSE) has been successfully applied across the software development lifecycle [1], including the areas of analysis and design, where Harman suggests that “the concept of evolutionary computation has effected virtually every area of software design, not merely as a metaphor, but as a realistic algorithm for exploration, insight and improvement” [2]. Thus given the success of such approaches in evolving models to solve problems in software engineering, what now is the role of the software engineer? On the face of it, SBSE has reduced the cognitive workload of problem solving for the engineer. Thus is the role of the software engineer now reduced to selecting the appropriate model representation(s) and fitness function(s), and simply let search take its course to arrive at superior candidate solution variants?

The notion of engineers solving software engineering problems by formulating model representations and corresponding fitness functions to drive search seems entirely logical. Indeed, as the home web-page of the First International Workshop on Combining Modeling with Search-Based Software Engineering (CMSBSE) reminds us, modeling plays a pervasive role in software engineering, providing the means to manage complexity through understanding. Software models and their cognitive properties are of crucial importance to their understanding by human beings. Thus it seems likely that formulating models of software and fitness functions corresponding to the required aspects of human understanding would be highly useful and powerful indeed. One attempt has been made to propose a framework for metrics for the quality of conceptual models [3]. However, it emerges that some of the metrics suggested appear to be specific to models of a particular phase of the software development lifecycle, while others appear specific to the domain of discourse. In addition, some metrics proposed rely on a high degree of model formality e.g. measuring Object Constraint Language (OCL) expressions, or UML state chart diagrams. Overall, these metrics could not be described as generally applicable to software engineering. Thus it seems likely that formulating models and fitness functions with sufficient generality for model-based search throughout software development is beyond current practice and will prove a non-trivial research undertaking.

Moreover, software models have limitations. Firstly, understanding models is subjective, varying in interpretation from engineer to engineer. For example, the understanding of a model by a novice engineer is likely to be very different from that of an expert. Secondly, as models are abstractions, they are useful but necessarily fragments of the whole software engineering problem. These limitations suggest that although model-based search can be very useful in tightly-defined problem situations, the role of the software engineer is crucial in interpreting the results of search techniques within the overall picture of the software development problem context.

Therefore, given that subjectivity and fragmentation of software model understanding cannot be eliminated, is there any way these might be exploited? Certainly, visualizing candidate solutions and then using the human as the “fitness function” for subjective evaluation has been successfully used in other fields e.g. arts and animation, music, virtual reality, image processing, data mining, cybernetics etc. and has been referred to as Interactive Evolutionary Computation (IEC) [4]. IEC has also been applied with some success in a number of design fields, such as cartoon facial characters [5], bridge design [6], manufacturing plant layout [7], ergonomic chair design [8] and urban furniture design [9]. In fact, there are various ways of involving humans in computational search. When known in advance, user preferences can be incorporated a priori to narrow and refine a search space. Conversely, such preferences can also be used to select solutions to be presented to the user after search. However, potentially the most powerful role for the user is to interact as search is conducted, enabling a dynamic search sensitive to subjective user evaluation. An early report of this approach attempted to provide insight and understanding in early lifecycle development [10]. A later report into early lifecycle software design combined objective quantitative fitness functions such as design coupling and cohesion with subjective qualitative evaluation of design elegance to discover candidate software design solutions that are not only superior with respect to cohesion and coupling, but also appear elegant and aesthetically pleasing to the software.
designer [11]. Recent work has suggested the concept of an interactive search-based software testing approach [12].

In CMSBSE, an interactive role for the software engineer during search addresses many of the cognitive challenges of manipulating software models. For example, as the model evolves, the software engineer has a sense of being included in a problem solving episode, especially as dynamic search is sensitive to the subjective evaluations of the software engineer. However, exploiting this software engineer interaction is an ongoing challenge because we must look for worthwhile forms of engineer evaluation in model-based search that are difficult to reliably capture otherwise. Some forms of evaluation might include, for example, aesthetics and elegance, or readability and comprehension. Steering interactive search by elegance and comprehension could potentially lead to models that promote reuse or perhaps discover model patterns. Other forms of evaluation might relate to simplicity or creativity. Not necessarily conflicting, evaluations of model simplicity and creativity might reflect an ‘Occam’s Razor’ approach to search while at the same time promoting exploration of the space to arrive at solutions novel to the engineer. Perhaps another worthy form of evaluation might relate to the “technical debt” of a model - an idea that “software engineers sometimes accept compromises in a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline), and that such compromises incur a "debt" on which "interest" has to be paid and which the "principal" should be repaid at some point for the long-term health of the project” [13]. The computational response to subjective engineer evaluation might be precise or fuzzy, and may enable engineers to observe the effect of constraint relaxation during search.

A further on-going challenge for interactive CMSBSE relates to computational learning during interactive search, and the possibility of self-adaptive model representation and fitness functions. For example, the importance of quality (non-functional) attributes of software systems is increasing as we move to huge scale systems and ‘systems of systems’. Although such quality attributes such as resilience to change, sustainability (‘greenness’), efficiency, security, adherence to standards, usability, etc. can be quantified, the understanding of such metric values can be arbitrary. Might not it thus be possible for interactive computational search to relate subjective engineer evaluation to quantitative values for quality attributes over many systems and thus evolve the fitness of the model and fitness functions? This would certainly promote the generality of models and their corresponding fitness functions. Following on from this, it also remains an on-going challenge to investigate how a ‘shared understanding’ of the evolving models between the software engineer(s) and search could be exploited to lead to a mutual learning environment for software engineer and computer.

To date, empirical studies of interactive SBSE generally have not been readily abundant in the literature. Their paucity is an on-going challenge to the field, although practical reasons may account for this. Recruiting an appropriately sized sample of software engineers to study that is representative of some meaningful segment of the software engineering community is a non-trivial task. Also, software engineers are all unique individuals with differing experiences, preferences and understanding of models. Furthermore, although software engineers can offer many insights during a problem solving episode, they are prone to non-linearity of focus over time. Consequently, rigorous experimental design relating to participant behavior and relevant statistical analysis are necessary. In addition, ethical approval of empirical investigations is required to ensure no harm befalls the human participants. Nevertheless, recent reports of interactive SBSE are emerging (e.g. [11], [12]) suggesting that the role for software engineers in interactive CMSBSE is not withering away at all, but rather presenting an important and open challenge to the field.

REFERENCES