Practitioner-Oriented Visualization in an Interactive Search-Based Software Test Creation Tool

Bogdan Marculescu*, Robert Feldt*†, Richard Torkar*†
*Blekinge Institute of Technology
School of Computing
Karlskrona, Sweden
†Chalmers and University of Gothenburg
Dept. of Computer Science and Engineering
Gothenburg, Sweden

***Abstract*—Search-based software testing uses meta-heuristic search techniques to automate or partially automate testing tasks, such as test case generation or test data generation. It uses a fitness function to encode the quality characteristics that are relevant, for a given problem, and guides the search to acceptable solutions in a potentially vast search space.**

From an industrial perspective, this opens up the possibility of generating and evaluating lots of test cases without raising costs to unacceptable levels. First, however, the applicability of search-based software engineering in an industrial setting must be evaluated.

In practice, it is difficult to develop *a priori* a fitness function that covers all practical aspects of a problem. Interaction with human experts offers access to experience that is otherwise unavailable and allows the creation of a more informed and accurate fitness function.

Moreover, our industrial partner has already expressed a view that the knowledge and experience of domain specialists are more important to the overall quality of the systems they develop than software engineering expertise.

In this paper we describe our application of Interactive Search Based Software Testing (ISBST) in an industrial setting. We used SBST to search for test cases for an industrial software module and based, in part, on interaction with a human domain specialist. Our evaluation showed that such an approach is feasible, though it also identified potential difficulties relating to the interaction between the domain specialist and the system.

I. Introduction

Software is often developed as only one component among many in complex, engineered systems. In such a situation, not all system developers can be expected to have software engineering expertise. Nevertheless, their domain knowledge is often critical in creating and selecting test cases. Search-based software testing can automatically create software test cases and thus potentially tap into a broader experience base by presenting these test cases to the developers, and allowing them to interactively select the most meaningful ones [1], [2], [3]. The human experts, henceforth referred to as “domain specialists” are, therefore, an integral part of software development, using their knowledge and experience to make those trade-offs.

The tests thus developed need to be represented in a way that system developers, domain specialists, and even users, can understand and that enables an informed selection to be made. This can be achieved by matching test representations to the domain, rather than to the traditional programming and testing languages. Adopting domain specific representations, thus, allows information to be presented in ways that are already familiar and avoids the added burden of adapting to new representations.

In this paper we investigate the effectiveness of such representations for interactive, semi-automated testing of software for embedded control systems developed by a Swedish systems engineering and manufacturing company. The company’s input was crucial in developing a prototype ISBST system, evaluating our assumptions, and learning from these efforts. Our contribution is one of the first deployments and evaluations of an Interactive Search Based Software Testing (ISBST) system in actual, industrial practice.

In Section II we present existing efforts with respect to interactive evolutionary search and related approaches, and discuss how our approach differs from them. Section III describes the industrial context that forms the basis of this work. Our approach is described in Section IV, together with a brief evaluation and a discussion on the lessons learned from this effort in Section V. Section VII concludes the paper.

II. Related Work

A term coined by Harman and Jones in 2001 [4], search based software engineering (SBSE) is the application of metaheuristic search techniques to software engineering problems, e.g. [5], [6], [7]. Search based software testing (SBST) is a branch of SBSE that deals with testing problems and has successfully been applied to several types of testing problems [1], [2], from object-oriented containers [8] to dynamic programming languages [9].

An important concept for search-based systems is that of fitness function. The fitness function can be seen as "the characterization of what is considered to be a good solution" [4]. The fitness function is used to select the best solutions in a population, and to guide the search towards good solutions.

Takagi defines Interactive Evolutionary Computation as "an EC that optimizes systems based on subjective human evaluation" [10]. This approach relies on human interaction to evaluate the solutions being developed by the interactive search system, allowing for situations where the choice of solution is dependent on "human preference, intuition, emotion and psychological aspects" [10]. The original paper refers to art and animation, graphics and image processing; in general applications where the evaluation of a candidate has a strong subjective component. A prominent example of this is Picbreeder [11], an online service where users evolve images, in a collaborative setting, using interactive evolution. The users' aesthetic preferences drive the evolution, not any objective goal.

Nevertheless, we think that this can be generalized to any type of candidate evaluation where not all the necessary information, e.g. domain specific knowledge, experience, background information, or intuition, can be modeled into the system or encoded in an automatic evaluation approach [12], [6]. From the perspective of the system, implicit knowledge can be seen as subjective evaluation, thus avoiding the need to duplicate existing expertise.

Tagaki also identifies the problem of human fatigue [10]. This is a problem that arises when a human user has to perform a large number of interactions with the system. This is an issue for any interactive system, since a human suffering from fatigue will not provide the level of analysis and decision making necessary to perform their duties appropriately. Therefore, a key element in an interactive system will not function properly and may even hinder the system's ability to evolve a good solution.

Search based approaches have already been used as exploratory tools, in situations where there is an incomplete knowledge of the search space. Feldt [6] describes genetic programming being used to explore potential designs for aircraft arresting system software, and identifies the importance of obtaining problem-specific knowledge early in the design process. Parmee et al. [13] introduces an Interactive Evolutionary Design System to support the early stages of design. It identifies the importance of capturing "design knowledge through extensive designer interaction".

SBST systems, like EvoSuite [14], require software engineering expertise to use. Adding interactivity to such a system in our context would require the domain specialists to acquire additional skills in software engineering, a process that would be costly in terms of time and resources.

Our system differs from previous approaches precisely in the issue of using the current domain specific representation as a base for interaction, while trying to shield the domain specialist from the software engineering specific details.

III. Industrial Context

Our industrial partner develops hydraulic and electronic products for off-highway vehicles and machinery. These products use embedded software for a variety of applications, from steering and transmission systems to sensors and displays, where software is an important but not the main component.

In addition, the company provide their customers with a software development tool, specifically designed to allow domain specialists to modify and develop their own embedded software, to be used with general purpose controllers. While the exact applications may vary greatly, in all these cases domain expertise outweighs in important software engineering expertise.

As a result, while software is an important component in their respective products, they focus their efforts and resources on other components.

In such a context, the quality of the resulting system depends on that of the software, but not exclusively so. Trade-offs may be needed, e.g. restrictions on software capabilities due to the need to use more robust and less capable hardware, that cannot be known ahead of time or may emerge during the design process.

A domain specialist has knowledge of the domain, and experience with the limitations and demands placed on the systems they are developing. This knowledge enables them to better assess the quality characteristics of the complete product.

A practical example is that of a mechanical arm: the hydraulic valves and electric motors are all controlled by a micro-controller. The software for this micro-controller is often developed ad-hoc for each application, so no generalized test cases can be developed.

In the example here, we are developing tests for a module of the micro-controller software. The system under test (SUT) is a filter that ensures that a signal, e.g. the input for a motor from the user or other modules, does not damage other components. It does so by ensuring that the signal does not exceed a given upper limit and attenuating any sudden changes. This filter is a relatively simple, but typical component: it is common enough to be included in the basic function library that is provided with the development tool mentioned above.

To gain a better understanding of the situation, we conducted discussions with our industrial partner and attended a training session for our industrial partner's and their clients' engineers.

Their approach is to provide domain specialists with training in the use of the software development tool, rather than trying to teach software engineers the domain concepts they would need. The software development tool they provide uses concepts that are already familiar to those working in the field, e.g. it expresses a component in terms of signals and operations on signals, rather than programming concepts. The trainees, in effect, create the types of models that they have been used to develop, and those models are used by the tool to generate code.

We have developed an Interactive Search Based Software Testing system, tailored to the specific need of the industrial partner and their development tool, and based on the type of software applications that they usually develop. The quality criteria we have used to evaluate this system and the resulting test cases are based on our discussions with our industrial partners, their current practices and their experiences.

The ISBST system itself is a web-based add-on, separate from the tool offered by our industrial partners. This means that, while the current SUT and application are specific to the company, the ISBST system can be extended to address other software problems. This will allow any findings to be generalized to a wider set of problems and situations.

IV. The ISBST System

A. Design of ISBST system

The ISBST system we have developed, described in more detail in [12], consists of a search-based software testing system, where the search objectives are selected by interaction with domain specialists.

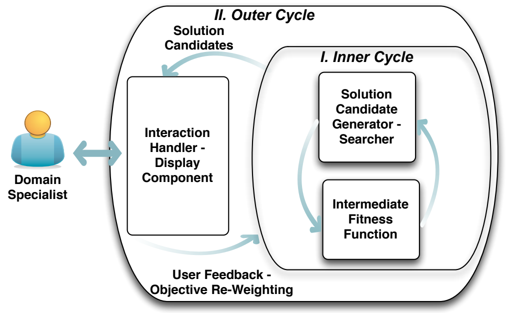
II. Outer Cycle
Solution Candidates
I. Inner Cycle
Solution Candidate Generator - Searcher
Interaction Handler - Display Component
Domain Specialist
Intermediate Fitness Function
User Feedback - Objective Re-Weighting

Fig. 1. Overview of the ISBST

The system searches for test cases that break, or are close to breaking, the limitations that are set as objectives: the upper limit, the number of found discontinuities in the signal, and the size of the signal. The first two match the desired functionality of the SUT in question, while the third favors the search for shorter test cases. A shorter test case takes less time to execute and is easier to understand and evaluate by a domain specialist.

The interaction allows domain specialists to use their experience to evaluate and rank the solutions developed by the ISBST system. To allow this interaction to take place in a meaningful way, the system uses representations that are relevant to the domain specialist and that allow for a fluid communication of the necessary information to them. It also allows for domain specific representation, separating the domain specialist from the minutiae of the underlying ISBST system.

This means that the specialists in question will continue to work with concepts and ideas familiar to them and do not need to develop software-specific skills to use the ISBST system to create test cases.

The ISBST system (Fig. 1) consists of two nested cycles.

The inner cycle contains the search based software testing system that forms the basis of our approach. The domain specialist guides the search indirectly, by selecting the quality criteria that they consider important at a given time and prioritizing them. The fitness function is then adapted to reflect those priorities.

The outer cycle handles the interaction with the domain specialist, including quality criterion selection and prioritization, and the visualization of the candidates.

On a more technical level, the inner cycle is the back-end of the system and has been developed in Ruby. It uses the differential evolution algorithm found in the FeldtRuby library to evolve the candidate solutions, and then a variant of Bentley's Sum of Weighted Global Ratios [15] method for evaluating them.

The outer cycle uses combination of html and javascript to display the candidate solutions to the domain specialist. The Data-Driven Documents library (D3) to provide a suitable graphical visualization for the candidate solutions.

As mentioned above, Takagi [10] identifies user fatigue as a main problem in using interactive evolutionary computation. He also proposes some methods of alleviating that problem.

In our system, the outer cycle handles any issues regarding the interaction with the domain specialist, including the prevention of user fatigue. To this end, some of the methods presented in [10] have been adopted. Only some of the large number of solutions being generated are displayed, with the selection being performed on the basis of the priorities the domain specialist has stated. Interaction events take place once every 500 optimization steps, to further alleviate the problem of fatigue.

B. *Use of the ISBST System*

The current ISBST prototype is used via a web interface. It allows the domain specialist to select from a list of objectives those that are relevant to them currently and to provide a prioritization by assigning the appropriate weight to each objective.

The weighting provided is used to dynamically develop the Intermediate Fitness Function (IFF) that will be used by the ISBST to create the first set of candidate solutions.

During the next interaction step, the domain specialist can evaluate the top ranked solutions from that set. An overview of the candidate solutions allows for an at-a-glance comparison (Fig 2), while individual views can provide additional information.

It is also during the interaction step that the domain specialist to adjust their selection and prioritization of the objectives by re-weighting them. Once all these activities are complete, the process of dynamically computing the IFF, developing a new set of candidate solutions and presenting the top ranked ones for evaluation is repeated until one or more satisfactory candidate solutions have emerged.

V. Empirical Evaluation

The empirical evaluation of the ISBST prototype has two goals: the wider one of investigating the applicability of ISBST for companies in this industrial domain, and the narrower evaluation of the interaction mechanisms chosen for the prototype. These goals constitute an initial evaluation, part of the wider goal of determining what is

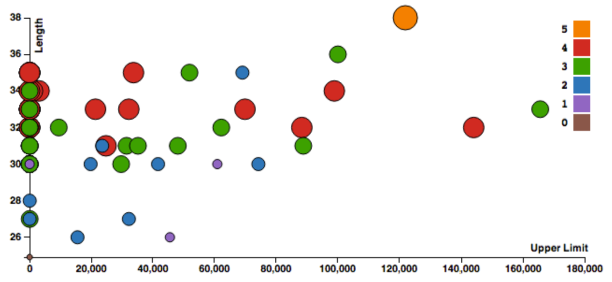

Fig. 2. An overview of the best available candidate solutions during an interaction. The diagram shows information regarding the score each candidate solution obtained regarding the three objectives that were used for the evaluation: Length of the test case (Y axis), the degree to which each test case approaches or even exceeds a set upper limit is given by the Upper Limit (X axis) score, and the number of discontinuities that were discovered (color). This diagram provides an at-a-glance way of comparing candidates. More information on each candidate is also available in the form of an individual view.

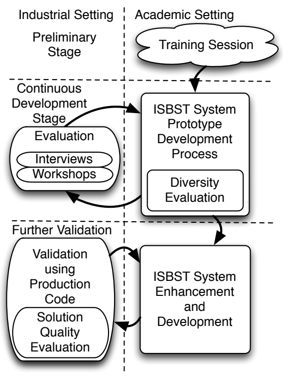
Industrial Setting
Academic Setting
Preliminary Stage
Training Session
Continuous Development Stage
Evaluation
Interviews
Workshops
ISBST System Prototype Development Process
Diversity Evaluation
Further Validation
Validation using Production Code
Solution Quality Evaluation
ISBST System Enhancement and Development

Fig. 3. An overview of the stages and empirical work of this study.

the level of quality of the tests found using this method. An overview of the evaluation method can be seen in figure 3.

The first goal, the applicability of ISBST in this industrial context, was evaluated by the degree to which the prototype system evolved solutions according to the search objectives and weightings set by the domain specialist.

The second goal, evaluating and improving the interaction mechanism used by the ISBST prototype system. The mechanism is that of dynamically developing the Intermediate Fitness Function (IFF) based on the objective selection and re-weighting. To evaluate this mechanism, we measure the degree to which the diversity of the resulting test cases is affected by the choice of fitness function.

The empirical evaluation consisted of two stages. In the first stage, the choice of interaction mechanism was validated in a laboratory setting. The second stage consisted of evaluating the ISBST prototype with the company's development and testing team. In practice, the two stages overlapped into a Continuous Development Stage (fig. 3), with new information being incorporated into the system.

The table I, shows a few of the strategies of use that were investigated as part of the empirical evaluation. The effect of each of the strategies on candidate population diversity can be seen in Figure 4.

The second stage consisted of workshops and interviews with the development and testing team. These workshops provided further validation for the interaction mechanism, as well as lessons regarding improvements that can be made to the system.

The workshops also resulted in a set of lessons learned, described in Table II.

Overall the results indicate that the mechanisms we have used to handle the interaction between the domain specialist and the ISBST prototype are useful and usable. The domain specialists responded well to the prototype and were extremely helpful both in evaluating it and in providing suggestions for improvements in subsequent versions.

On a more practical level, a number of the tests generated were successful in achieving the search objectives, i.e. identifying inputs that cause the system of exceed the maximum value set or cause the output signal to have discontinuities.

VI. Discussion

One medium term goal is to tap into the knowledge and experience of domain specialists to guide the ISBST system towards interesting and meaningful test cases, thus improving quality without leading to prohibitive costs. Another is to apply this concept in other domains that can benefit from automated test case development, yet where domain knowledge is vital. Further still, other phases of the software development process could also benefit from the combination of human insight and automated computing power.

In the long term, we hope to enable human inspiration and experience, things that cannot be captured in an automated system, to guide rather than limit software development. Such human specific contributions would, therefore, actively improve the solutions rather than being abstracted away as inconvenient or unpredictable.

In addition to this vision, however, some threats to the validity of this study must also be discussed.

The ISBST prototype system was developed for a specific company and is therefore limited by using a single set of procedures and relying on one source of information and experience. That said, the company develops a wide range of embedded software for a generic type of controller. Future studies will investigate the issue of generalizability further, but, based on current results, we feel that this approach shows considerable promise.

VII. Conclusions

This paper has presented an industrial application of search based algorithms. The Interactive Search Based Software Testing (ISBST) prototype system we have proposed and implemented uses the knowledge and experience of domain specialists to guide the search algorithms towards interesting solutions.

Initial results are promising, showing that the interaction between domain specialists and automated test case generation tools is a sound approach and can yield useful results. The same results also show the importance of the interaction mechanisms and of the information being provided to the domain specialists, in order to make their decisions.

On a higher level, the importance of dynamic validation, with company personnel and in an industrial context is apparent, as practical use of the prototype yielded more information that the preceding static validation efforts.

References

[1] P. McMinn, "Search-based software testing: Past, present and future," *Fourth International Conference on Software Testing, Verification and Validation Workshops*, pp. 153–163, 2011.

[2] 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. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2008.12.005

[3] R. Feldt, "An interactive software development workbench based on biomimetic algorithms," Gothenburg, Sweden, Tech. Rep. 02-16, November 2002.

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

[5] S. Xanthakis, C. Ellis, C. Skourlas, A. L. Gall, S. Katsikas, and K. Karapoulios, "Application of genetic algorithms to software testing," in *Proceedings of the 5th International Conference on Software Engineering and Applications*, Toulouse, France, 7-11 December 1992, pp. 625–636.

[6] R. Feldt, "Genetic programming as an explorative tool in early software development phases," in *Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering (SCASE '99)*. University of Limerick, Ireland: Limerick University Press, 12-14 April 1999, pp. 11–20.

[7] M. Harman, S. A. Mansouri, and Y. Zhang, "Search based software engineering: A comprehensive analysis and review of trends techniques and applications," Tech. Rep. TR-09-03, April 2009.

[8] A. Arcuri and X. Yao, "Search based software testing of object-oriented containers," *Information Sciences*, vol. 178, no. 15, pp. 3075 – 3095, 2008.

TABLE I
INTERACTION STRATEGIES USED TO INVESTIGATE THE PRESERVATION OF POPULATION DIVERSITY

Scenario	Description
A	Non-dynamic, multi-objective fitness function.
B	Focus in on one single objective that completely and consistently outweighs the others.
C	The objective under focus outweighs the others, but is changed at the last step.
D	A more balanced approach. One objective outweighs the others, but it is not the sole focus of attention.

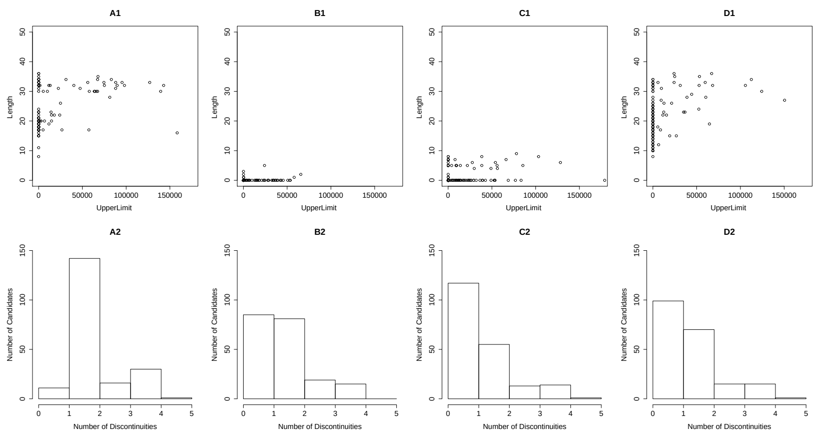

Fig. 4. Population diversity after a set of Interactive Objective Re-Weightings

TABLE II
LESSONS LEARNED AS A RESULT OF THE EMPIRICAL EVALUATION

Lesson	Description
1	**Understandability**. Providing clear, understandable, and accurate information to the domain specialist is the key element in establishing a meaningful interaction between them and the ISBST system.
2	**Variation**. A single developer may work on several systems in a short amount of time. In addition, there is a large amount of variation in terms of the types of systems developed within the same company.
3	**Individual Preferences**. Even when faced with similar systems, different individual domain specialists may have different preferences regarding how the interaction should take place and what information should be made available.
4	**Dynamic Validation**. Dynamically validating the system allowed us to identify potential problems in a timely manner and to develop a system that was relevant to our industrial partners and that could, in turn, benefit from their experiences.

[9] S. Mairhofer, R. Feldt, and R. Torkar, "Search-based software testing and test data generation for a dynamic programming language," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ser. GECCO '11. New York, NY, USA: ACM, 2011, pp. 1859–1866. [Online]. Available: http://doi.acm.org/10.1145/2001576.2001826

[10] H. Takagi, "Interactive evolutionary computation: fusion of the capabilities of ec optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275 –1296, sep 2001.

[11] J. Secretan, N. Beato, D. B. D Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley, "Picbreeder: evolving pictures collaboratively online," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. New York, NY, USA: ACM, 2008, pp. 1759–1768. [Online]. Available: http://doi.acm.org/10.1145/1357054.1357328

[12] B. Marculescu, R. Feldt, and R. Torkar, "A concept for an interactive search-based software testing system," in *Search Based Software Engineering*, ser. Lecture Notes in Computer Science, G. Fraser and J. Teixeira de Souza, Eds. Springer Berlin Heidelberg, 2012, vol. 7515, pp. 273–278. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33119-0_21

[13] I. C. Parmee, D. Cvetkovic, A. H. Watson, and C. R. Bonham, "Multiobjective satisfaction within an interactive evolutionary design environment," *Evolutionary Computation*, vol. 8, no. 2, pp. 197–222, 2000.

[14] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 416–419. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025179

[15] P. Bentley and J. Wakefield, "Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms," in *Soft Computing in Engineering Design and Manufacturing*, P. Chawdhry, R. Roy, and R. Pant, Eds. Springer London, 1998, pp. 231–240.