Objective Re-Weighting to Guide an Interactive Search Based Software Testing System

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*—Even hardware-focused industries today develop products where software is both a large and important component. Engineers tasked with developing and integrating these products do not always have a software engineering background. To ensure quality, tools are needed that automate and support software testing while allowing these domain specialists to leverage their knowledge and experience.**

Search-based testing could be a key aspect in creating an automated tool for supporting testing activities. However, domain specific quality criteria and trade-offs make it difficult to develop a general fitness function *a priori*, so interaction between domain specialists and such a tool would be critical to its success.

In this paper we present a system for interactive search-based software testing and investigate a way for domain specialists to guide the search by dynamically re-weighting quality goals.

Our empirical investigation shows that objective re-weighting can help a human domain specialist interactively guide the search, without requiring specialized knowledge of the system and without sacrificing population diversity.

***Keywords*-search based software testing, interactive search based software engineering, user centered, embedded software, industrial experience**

I. Introduction

Software is often developed as one part of a more complex system, by companies whose core competencies lie in other fields. Such companies often lack the software development and testing expertise needed to perform extensive software quality assurance, choosing instead to focus their efforts on maintaining and developing vital domain-specific knowledge. This reflects the reality that the quality of software-intensive products depends on a series of trade-offs, and software quality is just one concern among many. Increasing a company's focus to include software engineering and testing would incur significant costs, without guaranteeing a significant increase in the overall level of quality of the product in its entirety. An alternative to this approach is to use a pre-packaged software testing toolbox to enable domain specialists to focus on applying their experience and knowledge of the domain, rather than developing software testing skills. In this context we use "domain specialists" to denote systems engineers and other specialists in their own fields that have to develop, use and test software. The importance of knowing domain-specific constraints and trade-offs outweighs that of achieving proficiency in software testing.

It is in this context that interactivity becomes important. A pre-packaged software testing toolkit is difficult to develop and optimize before the specifics of the application become known. Moreover, the precise focal points of the testing process may change from one project to another or may vary in time within the same project, further emphasizing the importance of integrating domain knowledge into any effective testing tool.

This paper focuses on the problems met by domain specialists in testing software and proposes a solution to address, or at least alleviate, these problems.

In section II, we consider existing approaches to interactive evolutionary search, and discuss how our approach differs from them. Section III describes the industrial context and a system we use as a running example. Then, section IV describes in more detail the objectives of this study and the way the system was designed to meet those objectives. A closer look at the implementation of that design can be found in section V.

A brief empirical evaluation is described in section VI, with some of the threats to the validity of the study discussed as well. Section VII concludes.

II. Related Work

Search Based Software Engineering (SBSE) is a term coined by Harman and Jones in 2001 [1] to describe the application of metaheuristic search techniques to software engineering problems, e.g. [2], [3], [4]. The branch of SBSE concerned with testing problems is known as Search Based Software Testing (SBST) and has been applied to several types of testing problems [5], [6], from object-oriented containers [7] to dynamic programming languages [8]. However, there has been very few studies considering interactive SBSE.

Feldt [9] described an interactive development environment where tests are created as the engineer write the program code or refine the specification. The system used the interactions of the engineer to help guide the search but the effect on the fitness function was indirect. Feldt [3] and Parmee et al [10] considered the use of interactive search to explore engineering designs and better understand design constraints.

In a previous paper [11], we proposed a system that combines several of these concepts, e.g. interacting with the system once in a number of generations, rather than each generation [12]; and adds that of interacting with a ISBST system by means of allowing the human to modify the fitness function. The Intermediate Fitness Function (IFF) describes the current goal and contains all the information available at a given moment. As a result, it is reasonable to assume that it should be updated as that understanding changes or becomes more refined, or as new information becomes available.

III. Industrial Context

The approach presented in this paper was developed as a result of input from our industrial partner and is shaped by the context where they operate.

Our industrial partner develops products that involve embedded software, but where software is not the main consideration. As a result, knowledge of the domain and domain specific trade-offs has a greater impact on the overall quality of the product than expertise in software development and testing. This type of situation emphasizes the importance of extensive knowledge of the domain and experience with the context and limitations of the application being developed. These allow a domain specialist to better assess quality characteristics that the complete product must have, as opposed to the quality level of the software as a separate entity.

We will use a running example to better illustrate the challenges we faced and to clarify the approaches we used to address these challenges. The example is that of a control mechanism for a electric motor, e.g. powering a mechanical arm. Due to limitations of the motor itself and the potential for damage in what the mechanical arm is handling, an average filter is necessary to convert sharp changes in input into a smoother signal for controlling the motor. The example is based on a model filter provided by our industrial partner. The filter is relatively simple, but common enough to be included in the standard toolkit of commonly used components.

The system under test (SUT) is the aforementioned average filter. In this particular example, there are three major quality goals to be accomplished. First, since the filter's purpose is to smooth a signal for use as input in a motor, it is important that the output is free from discontinuities which might damage the motor itself, the arm or anything the arm might be handling at the time. Second, there are limits to what inputs are acceptable to the motor, for the same reasons. Last, it is important that an input signal is as short as possible, to enable any test case to be human readable and understandable and to allow assessment of the test case in a reasonable amount of time.

During the project we held discussions with our industrial partner and their clients and took part in one of the training sessions for developing software using a domain-specific tool.

The approach of selecting domain specialists and giving them training in developing software using a domain specific tool illustrates the relative importance of domain knowledge and experience compared to expertise in software development and software testing. This guided our approach toward developing a support tool for domain specialists rather than trying to capture domain-specific knowledge for the benefit of software engineers. In addition to domain knowledge and experience, the system must account for the fact that the domain specialists may come from different, albeit related, backgrounds and are developing different and quite disparate products.

As a result of these efforts, we have developed an Interactive Search-Based Software Testing (ISBST) system. This system searches for test cases that break, or come close to breaking, the quality goals stated above, under the guidance of a domain specialist.

IV. Objective and Method

As touched upon in the previous sections, the software in this context is developed by domain specialists rather than software engineers. The main problem identified as a result of our discussions with our industrial partner was that of creating test cases that cover a significant part of the input space.

Currently, test cases are created ad-hoc for each module, based on the developer's intuition and experience with similar modules in the past. While this allows past lessons to be incorporated into the development work, it also means that new problems are hard to identify.

To address this problem we propose an ISBST system that would benefit from the exploratory potential of search based techniques, while still benefiting from the experience and intuition that domain specialists rely on.

This study aims to answer the following research question:

RQ1: *How can interaction between a human domain specialist and a search based testing system be achieved?*

To answer this question we propose Interactive objective re-weighting as a means of interaction, especially in situations where:

- The domain specialist is not a software engineer and cannot be required to obtain expertise in software engineering.
- The domain specialist cannot be expected to evaluate a significant proportion of the candidates, due to the large number of candidate solutions as well as their complexity.
- Maintaining population diversity throughout the process is an essential part of the approach.

A secondary question is:

RQ2: *If such an interaction can be successfully achieved, what limitations or guidelines can be identified to ensure that population diversity is not negatively affected?*

To answer these questions, we have used a number of interviews and workshops, conducted with out industrial partner, as a basis to develop an SBST system that would enable the domain specialist to interact by Interactive Objective Re-Weighting (IORW).

The Interactive Objective Re-Weighting approach relies on the notion of "Interactive Fitness Function" or IFF. The IFF is a fitness function that can be dynamically modified during the course of the search, to better match the current understanding of the requirements on the system. The IORW is one means of achieving a workable IFF.

The IFF for a candidate j is computed as follows:

$$IFF(j) = \sum_{i=1}^{nObjectives} Weight_i * Value_{i,j} \tag{1}$$

where $IFF(j)$ is the fitness value of candidate j, $Weight_i$ is the current weight of the objective i, and $Value_{i,j}$ is the fitness value of candidate j measured by objective i. An objective k can be deselected from the computation by having $Weight_k = 0$.

Our approach is to use the IFF as a surrogate, evaluating a number of *n* optimization steps, between interaction events. An 'interaction event' is one interaction between the system and the domain specialist. It consists of the system displaying the current weighting and the best candidates according to that weighting, and of the domain specialist conducting a re-weighting if they feel it is needed.

This will help reduce the burden on the domain specialist without sacrificing population diversity and size. The exact value of *n* may vary from one system to another.

The IORW approach is presented in more detail in Algorithm 1.

```
set the IFF to default;
set currentStep = 0;
while acceptableSolution == false do
    if currentStep == interactionStep then
        begin Domain Specialist Interaction Step;
        solutions are displayed and evaluated by the
        domain specialist;
        if domain specialist accepts one of the
        proposed solutions then
            acceptableSolution = true;
        else
            acceptableSolution = false;
        end
        if domain specialist adjusts objective
        weighting to better reflect their goals then
            change the IFF to reflect the objective
            re-weighting;
        else
            IFF remains unchanged;
        end
        currentStep = 0;
    else
        perform optimization step with the current
        IFF;
        currentStep += 1;
    end
end
```

Algorithm 1: Dynamic modification of the IFF via IORW

The default setting for the IFF function is that of having the weights for all objectives as having an equal weight, i.e. $\forall j, Weight_j = 1$.

Thus, the domain specialists adjust the weights of the various objectives, based on their understanding of their relative importance and, in doing so, shape the IFF for the next set of optimization steps. The objectives are defined and presented in domain-specific ways, thus allowing domain specialists to dynamically generate a new IFF using only concepts that they are already familiar with and actively using. This can be

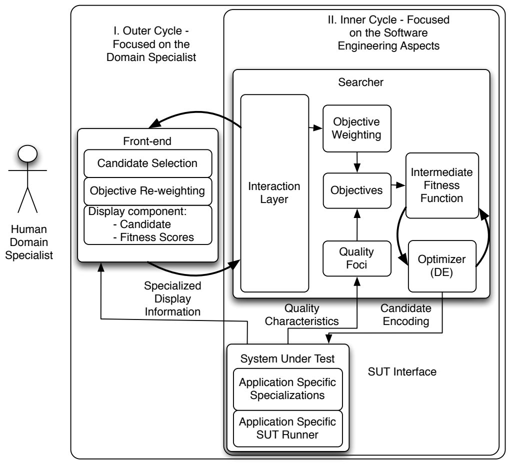
I. Outer Cycle - Focused on the Domain Specialist
II. Inner Cycle - Focused on the Software Engineering Aspects
Searcher
Front-end
Candidate Selection
Objective Re-weighting
Display component: - Candidate - Fitness Scores
Human Domain Specialist
Interaction Layer
Objective Weighting
Objectives
Intermediate Fitness Function
Quality Foci
Optimizer (DE)
Specialized Display Information
Quality Characteristics
Candidate Encoding
System Under Test
Application Specific Specializations
Application Specific SUT Runner
SUT Interface

Fig. 1. Overview of the ISBST system

done without requiring the domain specialist to achieve proficiency in software engineering in general, and in SBST in particular.

A. System Design

Figure 1 shows the design of the ISBST system. The system is split into two major components. The inner cycle deals with generating the candidates, running the candidates as inputs to the SUT and evaluating their fitness according to the IFF. The outer cycle handles the interaction with the domain specialist, i.e. displaying candidates and interpreting the feedback provided.

The design is shaped by the need to separate those components of the system that require interaction from the domain specialist, from those that can be fully automated. By achieving this separation, the exposure of the domain specialist to the underlying SBST system can be better controlled.

The system keeps the overall structure presented in [11], while making allowances for the increasing complexity inevitable in practical implementation.

a) The Outer Cycle.: The outer cycle is an interactive search based software system, where the domain specialist is presented with a set of candidates and, based on those candidates, they decide if the current objective weighting is appropriate. If it is not, then the objective weighting is changed to reflect the current understanding of the current quality needs.

The design uses some of the fatigue reduction concepts defined in [12]: e.g. requiring human interaction every n of generations rather than every generation, reducing the overall number of solutions that have to be evaluated by the human engineer, and focusing on aspects that do not require detailed evaluation of each individual solution. The outer cycle captures those interactions, while abstracting away the automated evaluations and removing the need for direct involvement with the technical details.

The human domain specialist guides the automated evaluation by providing the objective weighting that serve as input for dynamically generating the IFF. Moreover, this is done after the domain specialist is shown the fittest candidates, as evaluated by the current IFF. All this functionality is provided by the components of the outer cycle.

b) The Inner Cycle.: The inner cycle contains the search based software testing system developing the potential solution candidates, dynamically generates the IFF from the objective weights, performs the automated evaluation of candidates, and interfaces with the outer cycle and the SUT.

The interface to the SUT is concerned with the modules required to run the SUT for evaluation. Apart from information strictly necessary for interaction, the inner workings of the SUT are hidden.

The inner cycle functions as a traditional SBST system, with the `Searcher` generating the candidates, using the SUT to run those candidates, and evaluating them by means of the IFF.

V. Implementation

The outer cycle uses a combination of CoffeeScript, a variation of JavaScript aimed at simplifying development, and the Data-Driven Document (D3) library to display candidates and obtain feedback from the domain specialist. D3 is a JavaScript library for the manipulation and display of data in a browser. It allows visualizations to be dynamically created, without acting to change the data itself. Since the outer cycle is concerned with displaying the candidates, the combination of Coffescript and the D3 library is a good fit for the purpose.

The inner cycle and the SUT Interface are developed in Ruby. This was chosen for the relative ease with which it interacts with other applications, e.g. other SUTs, interfaces to simulators or interfaces to hardware test benches, enabling the ISBST to be more extensible overall.

First, a population of candidate solutions is generated by the `Searcher`. Each candidate is converted to a candidate object and run through the SUT or, by means of the SUT Interface, through the hardware test bench. Each candidate then receives a fitness score for each of the quality objectives that were selected. These scores, together with the respective weights, combine to form the IFF score, as discussed above.

c) Encoding the Candidate Solutions.: Our example is that of an average filter, that receives a signal describing the desired level of output and a number of steps to reach that level. In practice, our industrial partner uses a discrete encoding for each signal. Our system uses a Frame object to contain the input and output values at a given discrete step. Both are real numbers, with the outputs being computed by running the inputs though the SUT.

The candidate object consists of the ordered set of Frames, and also encapsulates the fitness values measured for each of the objectives.

When user interaction is required, the candidate objects are packaged into a JavaScript Object Notation (JSON) file and made available to the outer cycle. Search-related meta-information, e.g. fitness values for each objective, is included in the candidate object, and therefore available in the outer cycle. This enables the system to more easily adapt to changes in display requirements.

VI. Evaluation

This section presents an initial evaluation of the ISBST system presented above, and in particular of the mechanism of Interactive Objective Re-Weighting (IORW) used to handle the interaction with the domain specialist.

The criteria by which the success of this evaluation will be judged are as follows:

- A system can be developed that enables a domain specialist not familiar with SBST to guide a search using IORW. Successful use of IORW would prove the viability of the ISBST system we are proposing.
- The interaction between the domain specialist and the ISBST system can be shown to have an effect. In addition, identifying any potential differences between interaction strategies, would enable further studies to refine our understanding of the interaction, its benefits and limitations.

The evaluation uses the SUT described above, which is a Ruby implementation of the filter described in section III. The filter is a component of the standard library used by our industrial partner and offered to their clients. It was chosen for being a very simple, but common, component that is actively used in practice. For the purposes of this evaluation, a number of discontinuities were injected, to simulate faults in the software.

- Discontinuity (D). Measures the number of discontinuities found in a candidate. The aim is to maximize this objective.
- Upper Limit (UL). Given an upper limit for the input signal, this objective measures how close a given value is to that limit and, if the limit is exceeded, the difference between the limit and the actual value. The objective is to be maximized, as the goal is to find the signals that might exceed the maximum value.
- Length (L). For the purpose of this evaluation, we consider that a candidate should have close to a given length. As a result, candidates with lengths lower than *Min* frames or higher that *Max* receive a positive cost, while candidates with a length between the two values receive a negative cost. This objective is to be minimized.

By formulating the interaction in terms of objectives and their weights, the domain specialist interacts with representations and objectives that are familiar, and do not have to acquire additional software engineering skills.

An instance of IORW signifies that the domain specialist has re-evaluated their goals and priorities, either in light of new information becoming available from outside the system, e.g. changes in goals, domain specific constraints, or as a result of candidates noticed in the population of candidates being displayed.

In our example, a domain specialist starts their analysis of the SUT and their first priority is identifying discontinuities in the system. The second most important objective is the length of the candidate.

A set of interaction events were performed at regular intervals, once every 250 optimization steps. The weights are given as integers, from 1 to 5.

The set of interaction events in Table I shows an example of how a domain specialist can interact with the system, by specifying the relative importance of the respective objectives. In the example there, the first run is the default setting, with all objectives of equal weight. The domain specialist can choose to focus on one objective, e.g. Event 2, or try to find a more balanced approach, e.g. Event 3.

Through all the interaction events, the domain specialist evaluates the importance of each objective and the candidates they are shown, with the minutiae of the underly process being hidden from view.

Table II shows the mean fitness values of the displayed candidates for each quality objective.

An interactive objective re-weighting event, as expected, results in changes in the candidates selected for display to the domain specialist, as well as in changes to the IFF and the candidates that are generated. This increases the confidence that the re-weighting has a significant impact on the ranking of the candidates, as they are displayed to the domain specialist. Since the

TABLE I
EVENT SEQUENCE FOR INVESTIGATING THE IMPACT OF A SINGLE INTERACTION EVENT ON THE CANDIDATES. THE ABBREVIATIONS: EV - EVENT, D - DISCONTINUITY, UL - UPPER LIMIT, L - LENGTH

Ev	Description	Weights		
		D	UL	L
1	Describes the status at the beginning of the first interaction event. Initially, each objective is assigned an equal weight	1	1	1
2	The weight of the Discontinuity objective is raised to 5, signifying that the Discontinuity objective is about 5 times more important than the others	5	1	1
3	The weight of the Length objective is raised to 3. Finding a shorter candidate is more important than the upper limit, but less important than finding a discontinuity.	5	1	3
4	The relative importance of Length and Discontinuity are reversed. Finding a discontinuity is more important than the upper limit, but less important than finding a short candidate.	3	1	5
5	Return to the prioritization in step 3.	5	1	3

TABLE II
MEANS OBJECTIVE VALUES OF DISPLAYED CANDIDATES

Event	Discontinuity	Upper Limit	Length
1	2.333	13755	-6.2
2	3.6	12634	-0.333
3	3.8	5805	-0.533
4	2	3570	-8.866
5	3.333	15032	-2.266

ranking is based on the IFF, it is reasonable to claim that the re-weighting has an impact on that function and, through it, on the SBST system.

The issue of validity also needs to be discussed. The study presented here resulted from our efforts with one company, in one particular domain. To alleviate this problem a general SUT was selected and, by treating the SUT as a black box, the ISBST system is not tied to any particular implementation. These considerations indicate that the approach could be generalizable. To ensure such generalizability, however, further efforts are needed.

An additional issue is that the SUT used for the evaluation is a simple component. While the filter itself is simple, it cannot be considered quite a toy example: it is widely used and present in a toolbox of common components.

VII. CONCLUSIONS

In this paper we presented an Interactive Search-Based Software Testing system where the main method of interaction is that of objective re-weighting. The approach is a novel type of interaction based on our experiences with our industrial partner and their clients, and is based on their current testing practices and concepts. The approach does provide the flexibility and extensibility needed and allows domain specialists to interact with an ISBST system. The empirical evaluation indicates that the approach is viable; objective re-weighting can help a human domain specialist, unfamiliar with SBST or SE, interactively guide the search by adjusting their objectives alone.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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. [Online]. Available: http://www.dcs.kcl.ac.uk/technical-reports/papers/TR-09-03.pdf

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

[6] W. Afzal, R. Torkar, and R. Feldt, "A systematic review of search-based testing for non-functional system properties," *Information and Software Technology*, 2009.

[7] A. Arcuri and X. Yao, "Search based software testing of object-oriented containers," *Information Sciences*, vol. 178, no. 15, pp. 3075 – 3095, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020025507005609

[8] 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

[9] R. Feldt, "An interactive software development workbench based on biomimetic algorithms," Gothenburg, Sweden, Tech. Rep. 02-16, November 2002. [Online]. Available: http://www.cs.bham.ac.uk/~wbl/biblio/cache/http___drfeldt.googlepages.com_feldt_2002_wise_tech_report.pdf

[10] 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.

[11] 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

[12] R. Kamalian, E. Yeh, Y. Zhang, A. Agogino, and H. Takagi, "Reducing human fatigue in interactive evolutionary computation through fuzzy systems and machine learning systems," in *Fuzzy Systems, 2006 IEEE International Conference on*, 0-0 2006, pp. 678 –684.