**Emerging Topics in Software Engineering**

**UFCFCD-15-M**

**Student Number: 15027313**

**A Critical Review of Utilising Search-Based Software Engineering Techniques to Refactor Code**

**Word Count:**

# Introduction

Search-based software engineering (SBSE) is the name given to the search-based optimisation subfield of software engineering. It involves using metaheuristic search techniques to find a near-optimal solution, and will result in a decent solution but cannot guarantee that it is the very best solution. This essay will serve as a critical review of recent search-based refactoring (SBR) research articles.

“Software systems are subject to continual change and as they evolve to reflect new requirements, their internal structure tends to degrade. The cumulative effect of such changes can lead to systems that are unreliable, difficult to reason about, and unreceptive to further change.” (Harman and Tratt, 2007).

“The process of writing code can be compared to that of writing a paper or any other document – it starts with some ideas, a few drafts, and then a lot of time spent on refining and rewording, and is not something that should be rushed. Quality code takes a lot of time to write, and just like the writing of draft after draft will increase the author’s understanding of the topic area, code is written, changed, and changed again times and times over as the developers’ understanding of the system and the requirements expands. It is in the human nature to start small and build upward; in this case starting with code that works with limited functionality, expanding it to implement more functionality, and then refining it to optimise performance, readability, error handling, and so on.” (Birkehaug, 2016)

# Focus of critical review

Search-based refactoring has been used for many goals, mainly various kinds of optimisation issues. The focus for this essay will be the use of SBSE to locate possible areas of, as well as performing, refactoring of existing code bases, and any novel issues regarding this that need to be considered. Refactoring is the activity of improving the internal structure of code, while leaving the external structure unchanged. Search-based refactoring is about automatically discovering and performing useful refactorings, which has previously been a predominantly manual activity due to the difficulties of successfully locating and accomplishing this automatically. The following research questions are posed:

R1: Have there been any novel issues raised concerning using SBSE for refactoring in the near past?

R2: What is the current goals attempted to be met by utilising SBSE for refactoring?

The research articles located in the search laid a good foundation for locating the key researchers of SBSE, in particular SBR. Mark Harman and William Langdon from UCL are both two widely published and cited researchers in the SBSE area, even though Harman’s papers are represented in greater number than Langdon’s in this particular review. Marouane Kessentini is also the co-author of several papers, six of which referenced in this essay alone. He often occurs together with Mel Ó Cinnéide, Rim Mahouachi and Mohamed Mkaouer, and it appears that some of the researchers work together more than others, presumably based around interests and physical locations. The large number or citations may indicate that this is a topic on the rise, as table 1 lists citations for some of these authors since 2011.

Table 1: (Google Scholar, 2016a; 2016b; 2016c; 2016d)

|  |  |
| --- | --- |
| Author | Citations Since 2011 |
| Mark Harman | 9242 |
| William Langdon | 3597 |
| Marouane Kessentini | 459 |
| Mel Ó Cinnéide | 545 |

# Survey methodology

The principal method behind locating research papers was through the use of Google Scholar. The reason for choosing this particular search engine is that they display articles from a variety of different sources, and is a quick way to get an overview of papers as well as an indication of their recognition in the field, as they include a count of citations for each publication. While the citation count may indicate popularity and high-quality research, it should also be noted that these may be very low for newer publications, and it is thus difficult to judge how well a paper is received by these alone.

The first search done was “search based refactoring”, and did not include patents or citations. A brief skim of the abstract for the first 50 hits deemed the majority generally relevant, so as this review aims to cover state of art, the search was narrowed down to only include articles published since 2012, resulting in a publication range of approximately 4.5 years at the time of writing. The software engineering field moves very quickly and this range appeared to provide enough relevant publications to fulfil this work’s requirements.

The following list shows search hit number, titles and publication years of the first 20 hits, sorted by publication dates in descending order. The ones marked with an asterix are the ones that had full-text available and the titles in bold are those that were ultimately included in the review, with justifications to follow.

## Search results

|  |  |  |
| --- | --- | --- |
| # ft | Title | Year |
| 11 \* | **An experimental search-based approach to cohesion metric evaluation** | **2016** |
| 14 | [Book] Search-Based Software Engineering: 7th International Symposium, SSBSE 2015, Bergamo, Italy, September 5-7, 2015, Proceedings | 2015 |
| 20 | AutoRefactoring | 2015 |
| 10 \* | **On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach** | **2015** |
| 8 \* | **Search-based refactoring: Metrics are not enough** | **2015** |
| 1 \* | **Automated migration of build scripts using dynamic analysis and search-based refactoring** | **2014** |
| 3 \* | **High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III** | **2014** |
| 12 | On the use of machine learning and search-based software engineering for Ill-defined fitness function: a case study on software refactoring | 2014 |
| 13 \* | Search based software engineering for software product line engineering: a survey and directions for future work | 2014 |
| 18 \* | **[Keynote] Dynamic adaptive Search Based Software Engineering needs fast approximate metrics** | **2013** |
| 17 \* | Pareto-optimal search-based software engineering (POSBSE): A literature survey | 2013 |
| 7 \* | **Search-based refactoring detection** | **2013** |
| 9 | Search-based refactoring detection using software metrics variation | 2013 |
| 2 \* | **Search-based refactoring using recorded code changes** | **2013** |
| 16 \* | Dynamic adaptive search based software engineering | 2012 |
| 15 | Improving software security using search-based refactoring | 2012 |
| 4 | Search based software engineering: Techniques, taxonomy, tutorial | 2012 |
| 19 \* | Search-based model transformation by example | 2012 |
| 6 \* | **Search-based refactoring: Towards semantics preservation** | **2012** |
| 5 \* | **Search-based software engineering: Trends, techniques and applications** | **2012** |

## Selection criteria and excluded results

Based on the titles as well as the abstracts provided by Google Scholar, most of these 20 articles were seemingly relevant in the sense that they refer to search-based refactoring, and include one of the SBSE techniques in some way. A closer look did however reveal that this was not the case for all of them. Several excluded papers turned out to reference other potentially relevant papers, some of which were found in the search, but those that were not have not been included as a sufficient amount of papers passed the selection criteria.

* #4 (Harman *et al.,* 2012) was excluded, as it is a tutorial in the shape of a book chapter and not a research paper.
* #13 (Harman *et al.,* 2014) was also deemed irrelevant as it focuses on SBSE for software product lines, and only mentions refactoring in a generic listing of the possible applications of SBSE.
* #14 (de Oliveira Barros and Labiche, 2015) is also a reference to book and not an individual paper, and has been excluded.
* #16 (Harman, *Burke et al.,* 2012) is predominantly about dynamically adaptive software and not refactoring in the sense of only changing internal structure (as well as only mentioning refactoring a single time), and has been excluded too.
* #17 is a literature survey of papers using multi-objective search to find solutions, and only mentions refactoring in the titles of one of the surveyed works, which being published in 2007 also falls out of scope of this review.
* #19 does mention a plan of adapting their approach to other transformation problems including refactoring, but does not currently do so and has been excluded.
* #12, 15 and 20 were not available in full-text.

# Summary of reviewed papers

Following is a short summary of the ten papers that passed the selection and were available in full-text through a university account, as well as one (#8) that could fortunately be obtained directly from one of the authors.

1. “Automated migration of build scripts using dynamic analysis and search-based refactoring” (Gligoric *et al*., 2014), is using SBR to raise the abstraction level of the code, in order to assist the process of migrating build scripts.
2. “Search-based Refactoring Using Recorded Code Changes” (Ouni, Kessentini and Sahraoui, 2013) is using code changes recorded over time together with structural and semantic information in order to come up with more precise and efficient refactoring suggestions.
3. “High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III” (Mkaouer *et al*., 2014) proposes a scalable SBSE approach based on an evolutionary optimization method where the refactoring solutions are evaluated using 15 different quality metrics.
4. “Search-based software engineering: Trends, techniques and applications” (Harman, Mansouri and Zhang*,* 2012) provides a review and classification of SBSE literature, highlighting areas in need of more research.
5. “Search-based refactoring: Towards semantics preservation” (Ouni *et al*., 2012) focuses on finding an optimal refactoring sequence in order to minimise semantic errors while maximising quality improvements.
6. “Search-based refactoring detection” (Mahouachi, Kessentini and Cinnéide, 2013b) uses global and local heuristic search algorithms together with the code’s structural information to automate the detection of source code refactorings, using a manually revised version as a benchmark for the automatic refactoring, aiming to keep their metrics similar.
7. “Search-based refactoring: Metrics are not enough” (Simons *et al.,* 2015) recommends that future SBSE refactoring research should keep the human-in-the-loop in order to refactor code in a way that is helpful to the software engineers.
8. “On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach” (Boukdhir *et al.,* 2014) tackles the problem of optimising conflicting objectives by introducing a many-objective refactoring technique, evaluating refactoring solutions with a set of 8 distinct objectives.
9. “An experimental search-based approach to cohesion metric evaluation” (Cinnéide *et al.*, 2016) propose a search-based refactoring technique used to ‘animate’ metrics and observe their behaviour in a practical setting, in order to compare and discover how metrics relate to each other.
10. “Dynamic adaptive Search Based Software Engineering needs fast approximate metrics” (Harman, Clark and Cinneidez, 2013) discuss using metrics as fitness functions in order to search for sequences of refactorings and evaluate their effect on various metrics, with the goal of identifying metric relationships.

# Critical review

## Establishing previous state of art

As this review includes papers published in 2012 or later, the state of art from 2012 will be outlined based on a SBSE literature review, which was one of the articles selected in the search. Harman, Mansouri and Zhang (2012) identified that SBSE had been applied to refactoring, and that current research addressed their refactoring question “What is the best sequence of refactoring steps to apply to this system?”

They reported that there had been a dramatic increase in SBSE publications in the past five years, adding justification to restricting the publication scope to roughly 4.5 years for this new review. Their work acknowledged that there had been developments in the field resulting in several various approaches to using SBSE to automate refactoring, and that the SBR work they reviewed could be partitioned into two groups based on two main goals, as well as whether the approach was single or multi-objective. It appears that these two goals are still some of the main objectives of SBR at the time of writing, but also that variations of these have emerged.

Goal 1: Optimise the program

Goal 2: Optimise the applied sequence of refactoring steps

## Optimising

Partially falling under the second goal, Gligoric *et al.* (2014) used SBR to explore various sequences of refactorings to identify the shortest possible build script, and improved the runtime of the SBR by using the partial-order reduction technique, reducing the search space by applying a model-checking algorithm. In addition to aiming to reduce the number of refactoring steps, this is an interesting use of SBR with the aim to help repurpose code. This may be a recently arisen objective, as it was not reported by Harman, Mansouri and Zhang (2012).

Ouni, Kessentini and Sahraoui (2013) also focused on optimising the refactoring suggestions, but with a more common goal in mind than Gligoric *et al.* (2014). Their solution was to use a multi-objective optimisation approach in order to improve code quality, compared to what the observed results of techniques using only one or two objectives. By using records of previous code changes together with structural and semantic information, they used a search-based approach to improve the efficiency of new refactoring suggestions. The fitness function included design quality, semantic coherence and similarity to previously recorded refactorings, their weighing decided by a NSGA-II algorithm. The problem with this approach is that it requires a record of existing code changes for a system, but the future work they propose aims to remedy this by collecting refactorings from different systems and generalise their method to recognise possible refactorings based on both the refactoring type and the context. If they can achieve this, it will be a step towards a generalised refactoring tool, but current research only seem to achieve this on a small scale, for a limited set of systems

## The human aspect

Some of the papers identified in this work do indeed fall into the categories identified by Harman, Mansouri and Zhang (2012), but recent developments see the need of more categories in order to partition current state of art. For example, Simons *et al.* (2015) encouraged researchers to study ways to keep the human-in-the-loop for future SBR work. This may be perceived as backpedalling in a field that is generally concerned with automating tasks that historically have been primarily human-centric, but is actually fitting for the current state of automated refactoring.

Writing code is still at a stage where the human is incredibly important and required to both understand and maintain the code base. Discovering a way of bringing the software engineers back into the loop in an ordinarily increasingly automated task, to make decisions based on personal opinions, may ultimately result in solutions that more beneficial at this stage. Many software engineers are notoriously sceptic to automated refactoring, and the ability to imposing personal opinions on this may increase the acceptance rate of refactored code.

Should the creation of software ever become a completely or primarily automated task it may be advantageous to exclude the human aspect, but for now, they are arguably necessary. If this paper is to be categorised by goal like in the aforementioned review, the goal could fall under tailoring refactoring output to engineer preferences. While some may say that this falls under the goal of optimising the program, publications on using SBR to optimise a program are generally concerned with performance metrics and not the subjective opinions of the developers, meaning that Simons *et al.* (2015) are part of raising some important, fairly uncharted issues.

# Conclusions and future directions

Search-based software engineering is a field that has gained a lot of popularity in this century, yielding some promising products, but is still in the early stages of making a huge impact on refactoring. Judging by current research, a generalised SBR tool that revamps entire systems is still far away, even though steps are made in the right direction.

To conclude, there are several issues regarding SBR that researchers should address. The issue of the human-in-the-loop raised by Simons *et al.* (2015) is an important issue, as software engineers have always been vary of using tools to change their code. Research into how to include developers in an automated refactoring process could result in an increased approval of automated refactoring tools in the developer community, potentially resulting in more manageable code bases and less time spent on maintenance.

Several of the reviewed papers have brought up the issue of multi-objective refactoring, and while there have been some promising results, there is still a lot of work to be done in this field. There is little solid research done on the relationships between code metrics, and even less on how to factor this into refactoring choices. Searching for ideal, achievable sets of quality or performance metrics tailored to different needs, to be used as fitness functions for automated refactoring, is one of the potential research fields regarding this.

The use of records of previous refactorings in order to automate future refactorings is a very interesting topic in need of more research. With the current widespread use of version management software such as GitHub or SVN, structured use of these could make collecting data for this simpler, if paired with a system able to analyse the change history and deduct likely changes. If all refactoring was done in individual commits that were properly categorised, it could be very rewarding to analyse this data in order to build up a database and a system able to suggest or perform similar refactorings.

# REFERENCES

Friedberg, R. (1958) A learning machine: Part 1. IBM J Res. Dev. 2(1), 2–13

Friedberg, R., Dunham, B. and North, J. (1959) A learning machine: Part 2. IBM J. Res. Dev. 282–287

Harman, M. and Tratt, L. (2007) Pareto optimal search based refactoring at the design level. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation.* New York: ACM, pp. 1106-1113

## Articles and books

Amal, B., Kessentini, M., Bechikh, S., Dea, J. and Said, L.B. (2014) On the use of machine learning and search-based software engineering for Ill-defined fitness function: a case study on software refactoring. In: *Search-Based Software Engineering* [online]. Springer, pp.31-45.

Cinnéide, M.Ó., Moghadam, I.H., Harman, M., Counsell, S. and Tratt, L. (2016) An experimental search-based approach to cohesion metric evaluation. In: *Empirical Software Engineering* [online]. pp.1-38.

de Oliveira Barros, M. and Labiche, Y. (2015) *Search-Based Software Engineering: 7th International Symposium, SSBSE 2015, Bergamo, Italy, September 5-7, 2015, Proceedings* [online]. Springer.

Ghaith, S. and Cinnéide, M.O. (2012) Improving software security using search-based refactoring. In: *Search Based Software Engineering* [online]. Springer, pp.121-135.

Gligoric, M., Schulte, W., Prasad, C., Van Velzen, D., Narasamdya, I. and Livshits, B. (2014) Automated migration of build scripts using dynamic analysis and search-based refactoring, In: *ACM SIGPLAN Notices* 2014, ACM, pp. 599-616.

Harman, M., Burke, E., Clark, J.A. and Yao, X., 2012. Dynamic adaptive search based software engineering, In: *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on 2012*, IEEE, pp. 1-8.

Harman, M., Clark, J. and Cinneidez, M.O. (2013) Dynamic adaptive Search Based Software Engineering needs fast approximate metrics (keynote), In: *Emerging Trends in Software Metrics (WETSoM), 2013 4th International Workshop on 2013*, IEEE, pp. 1-6.

Harman, M., Jia, Y., Krinke, J., Langdon, W.B., Petke, J. and Zhang, Y. (2014) Search based software engineering for software product line engineering: a survey and directions for future work, In: *Proceedings of the 18th International Software Product Line Conference-Volume 1* 2014, ACM, pp. 5-18.

Harman, M., Mansouri, S.A. and Zhang, Y. (2012) Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* [online]. 45 (1), pp.11.

Harman, M., McMinn, P., De Souza, J.T. and Yoo, S. (2012) Search based software engineering: Techniques, taxonomy, tutorial. In:*Empirical Software Engineering and Verification* [online]. Springer, pp.1-59.

Kessentini, M., Sahraoui, H., Boukadoum, M. and Omar, O.B. (2012) Search-based model transformation by example. In: *Software & Systems Modeling* [online]. 11 (2), pp.209-226.

Mahouachi, R., Kessentini, M. and Cinnéide, M.Ó. (2013a) Search-based refactoring detection using software metrics variation. In: *Search Based Software Engineering* [online]. Springer, pp.126-140.

Mahouachi, R., Kessentini, M. and Cinnéide, M.Ó. (2013b) Search-based refactoring detection, In: *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation 2013*, ACM, pp. 205-206.

Mkaouer, M.W., Kessentini, M., Bechikh, S., Cinnéide, M.Ó. and Deb, K. (2015) On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. In: *Empirical Software Engineering* [online]. pp.1-43.

Mkaouer, M.W., Kessentini, M., Bechikh, S., Deb, K. and Ó Cinnéide, M. (2014) High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III, In: *Proceedings of the 2014 conference on Genetic and evolutionary computation* 2014, ACM, pp. 1263-1270.

Ouni, A., Kessentini, M. and Sahraoui, H. (2013) Search-based refactoring using recorded code changes, Software Maintenance and Reengineering (CSMR), In: *2013 17th European Conference on 2013,* IEEE, pp. 221-230.

Ouni, A., Kessentini, M., Sahraoui, H. and Hamdi, M.S. (2012) Search-based refactoring: Towards semantics preservation, In: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on* 2012, IEEE, pp. 347-356.

Santos Neto, Baldoino Fonseca dos, Ribeiro, M., Silva, V.T.d., Braga, C., Lucena, Carlos José Pereira de and Costa, E.d.B. (2015) AutoRefactoring. In: *Expert Systems with Applications: An International Journal* [online]. 42 (3), pp.1652-1664.

Sayyad, A.S. and Ammar, H. (2013) Pareto-optimal search-based software engineering (POSBSE): A literature survey, In: *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on* 2013, IEEE, pp. 21-27.

Simons, C., Singer, J. and White, D.R. (2015) Search-based refactoring: Metrics are not enough. In: *Search-Based Software Engineering* [online]. Springer, pp.47-61.

## Web sites

Google Scholar (2016a) *Mark Harman – Google Scholar Citations* [online] Available from: <https://scholar.google.co.uk/citations?hl=en&user=IwSN8IgAAAAJ> [Accessed 19.04.2016]

Google Scholar (2016b) *W B Langdon – Google Scholar Citations* [online] Available from: <https://scholar.google.co.uk/citations?user=O5cSyYMAAAAJ&hl=en> [Accessed 19.04.2016]

Google Scholar (2016c) *Marouane Kessentini – Google Scholar Citations* [online] Available from: <https://scholar.google.co.uk/citations?user=5oW_MA8AAAAJ&hl=en> [Accessed 19.04.2016]

Google Scholar (2016d) *Mel Ó Cinnéide – Google Scholar Citations* [online] Available from: <https://scholar.google.co.uk/citations?user=Cr8S0BwAAAAJ&hl=en> [Accessed 19.04.2016]