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: 3848

1 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. 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. 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 development of software is generally not a streamlined process, but one that emerges and evolves over time. Similarly to how writing numerous drafts will increase an author's understanding of their topic and goals, code will be written and rewritten multiple times before it reaches a high standard and proficiently fulfils its intended purpose.

At the beginning of a project the software engineers will have limited understanding of the proposed software, but as their understanding of the problems at hand develops, code is expanded on to gradually implement more functionality, and then ideally optimised to improve performance, structure, error handling, and so on. This is one of the reasons why refactoring may be considered a natural part of the code writing process, while it for non-developers seems unnatural to spend time redoing and perfecting work that has already been done.

2 Focus of critical review

Search-based refactoring has been used for many goals, mainly various kinds of performance 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 relating to this that need to be considered. 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 performed in section 3 laid a good foundation for locating the key researchers of SBSE. Mark Harman and William Langdon are both two widely published and cited researchers in the SBSE area, 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, who also took part in several of the referenced papers.

Table 1: Citation counts by author (Google Scholar, 2016a; 2016b; 2016c; 2016d)

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

Table 1 lists citations for some of the aforementioned authors since 2011, and it appears that this may be a topic on the rise. Over 9000 citations in the past 5 years by the most cited author, his papers are no doubt widely recognised and esteemed by the community.

3 Survey methodology

The 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. It eventually turned out that several of the papers that were included in this review did not have particularly high citation numbers, but they have only been published for roughly 4.5 years at the most, and belong to a topic that appears to be on the rise.

The search term for the first article search was “search based refactoring”, and did not include the patents or citations options that Google Scholar offer. 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. Further hits were not considered, as the relevance of the papers listed after the first 20 appeared to decrease rapidly. The entries marked with an asterix are the ones with full-text available, and the titles in bold are those of the papers that were ultimately included in the review, with justifications to follow below the table.

3.1 Search results

Table 2: 1-20 search results for “search based refactoring” from Google Scholar (2016e)

#	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**

3.2 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 at least one of the SBSE techniques in conjunction with this in some way. A closer look did however reveal that this was not the case for all of them. Several papers were excluded as they turned out to merely reference other potentially relevant papers, some of which were indeed found in the search, but those that were not have not been included as a sufficient amount of papers passed the selection criteria based on the aims of this review.

- #4 (Harman, McMinn *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 too been excluded.
- #17 (Sayyad and Ammar, 2013) 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 (Kessentini *et al.,* 2012) 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 (Amal *et al.,* 2014), #15 (Ghaith and Cinnéide, 2012) and #20 (Santos *et al.,* 2015) were not available in full-text.

4 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), uses 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) uses 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.
5. "Search-based software engineering: Trends, techniques and applications" (Harman, Mansouri and Zhang*,* 2012) provides a comprehensive review and classification of SBSE literature, identifying areas in need of more research.

6. "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 code quality improvements.
7. "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.
8. "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.
10. "On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach" (Mkaouer *et al.,* 2015) take on the problem of optimising conflicting objectives by introducing a multi-objective refactoring technique, evaluating refactoring solutions with a set of 8 distinct objectives.
11. "An experimental search-based approach to cohesion metric evaluation" (Cinnéide *et al.*, 2016) proposes a search-based refactoring technique used to 'animate' metrics and observe their behaviour in a practical setting, in order to compare and discover relationships between various metrics.
18. "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.

5 Critical review

5.1 Establishing previous state of art

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

They also 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.

Goal 1: Optimise the program

Goal 2: Optimise the applied sequence of refactoring steps

In addition to this, they observed a partitioning around 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.

Under refactoring, Harman, Mansouri and Zhang (2012) have listed no papers published after 2008, but they also recognise that their review was written before SBSE has become mainstream and may not thoroughly capture the new trend. They did however identify that SBSE was moving from single to multi-objective techniques, including the refactoring area.

5.2 Optimisation

Partially falling under the second goal, Gligoric *et al.* (2014) used SBR to explore various sequences of refactorings in order to identify the shortest possible build script. They 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 commonly observed goal in mind than Gligoric *et al.* (2014). Their solution was to use a multi-objective optimisation approach in order to improve code quality to a larger degree than 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, elements whose weighing was decided by a NSGA-II algorithm.

While this approach may seem promising, the problem concerning generalisation is that it requires a record of existing code changes for a system. The future work they propose aims to remedy this by collecting refactorings from different systems and generalise their method to identify possible refactorings based on both the refactoring type as well as the context of the code. If they can achieve this, it may be a big step towards a more general-purpose refactoring tool. However, current research only seem to achieve this on a small scale, for a limited type of systems to which they have been tailored.

5.3 Metrics as fitness functions

The work described by Mkaouer *et al*. (2014) again falls under a generic program optimisation goal. They proposed a scalable SBSE approach where they analyse refactoring solutions using 15 different quality metrics, claiming that their NSGA-III approach represents the new state of art in fully automated refactoring. Mkaouer was also part of similar research using the same NSGA-III approach with eight objectives (Mkaouer *et al.*, 2015), applied to a real-world application, and so it appears that Harman, Mansouri and Zhang (2012) were right to predict a rise in research done with multi-objective approaches.

Ouni *et al.* (2012) focused on both maximising the program structure as well as minimising domain semantic errors for their suggested refactoring sequences, but lack the breadth of objectives covered by Mkaouer *et al*. (2014) and Mkaouer *et al.* (2015) and their multi-objective approaches. However, while the multi-objective approach is reported to have a higher code smell detection rate in these studies than approaches using less metrics such as NSGA-II algorithm used by Ouni *et al.* (2012), it also has potential downsides.

It is not always a code smell can be represented by a set of quality metrics like NSGA-III method requires, or that it results in a code change the a software engineer will find more favourable than the original code. It may work well for optimising the performance of an existing code base, but that does not necessarily mean that the result will be easier to expand on. This is something Simons *et al.* (2015) try to remedy by bringing the engineers back into the refactoring process, which is reviewed under the "The human aspect" section below.

Harman, Clark and Cinneide (2013) based their work around metric relationships, more specifically on using metrics as fitness functions in order to search for refactoring sequences, and like Cinnéide *et al.* (2016), aimed to identify their relationships.

Cinnéide *et al.* (2016) established that there is generally a low understanding of the relationships between metrics and a lack of an established metric comparison methodology, which is the reason behind their novel SBR based technique used to 'animate' metrics in order to observe how they behave. They tested it on ten real-world Java systems, performing a variety of search techniques in order to achieve different goals such as increasing or decreasing metric agreement.

Metrics are increasingly used to base refactoring decisions on, for example such as seen in the articles by Mkaouer *et al*. (2014) and Mkaouer *et al.* (2015). They appear to be the topic of numerous pieces of research, but the current state of art appears to agree that there is no one metric that can be used to successfully rate a software system. Cinnéide *et al.* (2016) concluded that it is perhaps impossible to combine current cohesion metrics into a single metric.

5.4 Refactoring detection

Mahouachi, Kessentini, and Cinnéide (2013b) introduced a SBR approach based on reconstructing a revised version by representing a new version by a series of refactorings performed on the original version. They hypothesise that when their metric profiles converged, the refactoring steps taken to create this new, refactored version would converge with the steps taken to refactor the original revised version. This is a more novel way of making the refactoring take on an automatic yet human-like method. Suppose that this could be generalised and taught to learn a software engineer's coding style, it would then make refactoring decisions similar to the ones the engineer would, and consequently produce code structured similarly to what they would produce, thus resulting in code that makes sense to the engineer and is easier to maintain further.

5.5 The human aspect

While some of the papers identified in this work fall into the two aforementioned goal-based categories identified by Harman, Mansouri and Zhang (2012), recent developments see the need of more categories in order to partition current state of art properly.

For example, Simons *et al.* (2015) encouraged researchers to study ways to keep the human-in-the-loop for future SBR work. They claim that while some refactoring metrics are more popular than others, there is little correlation between the subjective opinion of software engineers and these sought-after metrics. Thus, they find no obvious way that software can successfully assess all aspects of perceived software quality, resulting in the need for human influence in refactoring decisions. 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 in fact fitting for the current state of automated refactoring.

The process of writing code is still at a stage where the human is incredibly important, required to both understand and maintain the code base. Discovering a way of bringing the software engineers back into the loop in an task that is becoming increasingly automated, in order to influence decisions based on personal opinions, may ultimately result in solutions that are more beneficial with the way things stand. Many software engineers are notoriously sceptic of automated refactoring, and the ability to impose personal opinions on the process may increase the acceptance rate and extent of refactorings.

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 the paper by Simons *et al.* is to be categorised by goal like in the aforementioned review by Harman, Mansouri and Zhang (2012), 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 essential, uncharted, and potentially very significant issues.

6 Conclusions and future directions

Search-based software engineering is a field that has gained a lot of popularity in this century, resulting in some promising advances, but is still in the early stages when it comes to making an 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, and answer the research questions, there are several new and old issues regarding SBR that researchers should address. The issue of the human-in-the-loop raised by Simons *et al.* (2015) is an important, novel issue, as software engineers have always been vary of using tools to change their code. Research into how to include developers in an otherwise automated refactoring process can result in an increased approval and usage of automated refactoring tools in the developer community. The desired outcome of this is more manageable code bases and less time required to be spent on maintenance, resulting in more time available for value-adding work such as new or improved features, and overall better quality systems.

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 methods including this in refactoring choices.

The discovery of achievable, model sets of quality or performance metrics, tailored to the needs of various types of systems, is a topic of research required to support this. These sets could be used as fitness functions as part of an automated refactoring system, providing choices based on the most favoured and useful metrics for a particular system in order to fine-tune performance-critical code.

The use of records of previous refactorings in order to support and automate future recommendations is also a topic in need of more research. With the currently widespread use of version control software such as GitHub or SVN, collecting refactoring samples would not necessarily prove an unmanageable challenge. Structured use of VCS could make this simpler if all refactoring was to be done in individual, properly categorised commits. Analysing this data in order to build a database of refactorings, as well as a system able to suggest or perform similar refactorings, would lay remarkable groundwork for a general-purpose refactoring tool.

7 REFERENCES

7.1 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.

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

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.

7.2 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]

Google Scholar (2016e) *Search based refactoring – Google Scholar Citations* [online] Available from: https://scholar.google.co.uk/scholar?start=0&q=search+based+refactoring&hl=en&as_sdt=1,5&as_ylo=2012&as_vis=1 [Accessed 16.04.2016]