Turkish Online Journal of Qualitative Inquiry (TOJQI)
Volume 12, Issue 7, July 2021: 3726-3742

Search-Based Approaches for Software Modularization/Re-modularization

Divya Sharma[a], Dr. Ganga Sharma [b]

[a] Department of Computer Science, Ph.D. Scholar, GD Goenka University
[b]Department of Computer Science, Assistant Professor, GD Goenka University

Abstract

Search-based Software Engineering helps in providing efficient decision support, and well as certain insights as per the requirement. The theory of the concept is important to understand because these approaches could be applied to every SDLC phase (Software Development Life Cycle Model) from a specification requirement point of view to its maintenance as well as operations. Search issues like solving SE issues in search space implies to find an ideal solution. The complete distribution of techniques for the prediction of defects is detailed further. A total of 77.5% of the studies are done on different classification methods, where 1.5% is focussed on association and clustering methods, and lastly, 14% analysis and studies are done on the different estimation techniques. Additionally, nearly 35.21% of the conducted research makes use of private datasets, whereas 64.79% of the conducted studies make use of public datasets. To predict software defects, nineteen distinct techniques were implemented. Seven of the most used techniques are recognized in the forecast of software defects from the nineteen techniques. However, there does exist numerous researches detailing the use of the search-based method to predict the maintainability as well as the change in proneness. It was discovered that application results when the search-based methods were followed for defect prediction and effort estimation were highly encouraging. Hence the motivation for this study was based on which the associations of the results would help the researchers and practitioners to get a set of guidelines and it would help them in making better choices of using these search-based methods in its specific conditions.

Keywords: SBSE (Search-Based Software Engineering), Modularization, Re-modularization, Metaheuristic Algorithms.

1. Introduction

SBO's interest in SE has resulted in enhanced interest in other types of SE optimization not necessarily based on a “search”. In literature, the word “SBSE” was commonly used for any type of optimization where a problem domain originates from SE and a solution includes optimization as per some well-defined concept of fitness [3].

Metaheuristic search methods like genetic algorithms along with several well-known strategies are used in SBSE to address a broad range of problems related to the software. These difficulties can range from (but not restricted to): project planning, maintenance, inverse engineering, and understanding of source code, refactoring of source code to repairing of program. [2,11]

At present, SBSE's biggest application region is software testing where there are SBSE methods that can create, enhance, and optimize test suites.

We describe a significant reorganization in the architecture of the system as re-modularization, with the main objective of enhancing its inner quality and thus by not adding fresh characteristics or fixing bugs. In a maximum of cases, structural aspects, such as static dependencies between architectural entities, guide re-modularization. For instance, a common suggestion is that the cohesion must be as high as possible and the coupling must be less that is manually followed or by using the re-modularization and semi-automatic tools. Nevertheless, the reality is that there is no strong consensus on what constitutes a healthy architecture. Architectural quality is a subjective notion that with standard quality metrics is hard to assess.

For example, recent work started to question the structural cohesion/coupling dogma, stating that “coupling and cohesion do not seem to be the dominant driving forces when it comes to modularization”.

Other research showed that structural cohesion metrics are generally divergent in evaluating the same refactoring actions. Semantic clustering is a strategy based on data collection and clustering methods to obtain sets of comparable classes according to their vocabulary in a scheme. Semantic clustering is a strategy based on data collection and clustering methods to obtain sets of comparable classes according to their vocabulary in a scheme [4,15,28,35,51].

Software Engineering can be split into two types: First, Black Box Optimization, which is a typical issue of combinatorial optimization. Second, white box issues in which source code activities need to be considered. The most widely used optimization and search techniques used are Local Search, Stimulated Annealing (SA), Genetic Algorithms (GAs), and Genetic Programming (GP), Hill Climbing (HC), Greedy Replant Algorithm, Linear Programming (LP) techniques, and Integer Linear Programming (ILP) [1,7]. Miller and Spooner [1976] revealed the earliest attempt to apply optimization to a software engineering issue. The term SBSE was first used by Harman and Jones in [2001].

This research also discusses the efficient experimental configurations and methods used in literature to use search-based techniques for SEPM assignments [18,19]. When the software engineer isolates the subsystem, consideration is given to the software structure, regardless of what is relevant to his work. The quality of the graph partition is evaluated by the approach since it represents the software structure and uses heuristics to navigate all possible graph partitions through the search space. The issue can be solved by several possible heuristic methods and are examined in this article by taking the referred documents. Software clusters are independent of any programming language and we need the source code analysis tool in which a direct graph can be acquired from source code to accomplish this.

Low coupling is the sign of a well-designed software system. The notion of software cohesion was described by the person who defined it as the degree to which a module's inner content is linked [6,17,23,30]. When the software engineer isolates the subsystem, consideration is given to the software structure, regardless of what is relevant to his work.

The quality of the graph partition is evaluated by the approach since it represents the software structure and uses heuristics to navigate all possible graph partitions through the search space. The issue can be solved by several possible heuristic methods and are examined in this article by taking the referred documents. Software clusters are independent of any programming language and we need the source code analysis tool in which a direct graph can be acquired from source code to accomplish this. Low coupling is the sign of a well-designed software system. The notion of software cohesion was described by the person who defined it as the degree to which a module's inner content is linked [6,17,23,30].

2. Methods

This phase includes giving the study questions clear responses [14,19,32]. In this research, a total of three reviewers analyzed all of the collected data separately. The opinion they help majorly was following a consultant as an external specialist in the event of inconsistent views. We explain the information of these three phases in this chapter as it is related to the present topic of analyzing different analyses and studies which are also reported in detail while reviewing the literature [7,19,23,29]. Systematic literature review methods are described by the guidelines taken from Kitchenham [22,27,39]. The major perspective was to analyze and assess the secondary research studies commonly called systematic literature reviews (SLRs).

The set of questions that are to be used in the research must be listed with a clear objective in the mind of the researchers which can help in guiding the whole investigation process and the methodology of the research to be followed [11,19,44]. These questions were based on an intention on which the different studies are discovered to analyze a phenomenon or a precise topic.

In addition to this, the protocol followed for the review also comprised of a search strategy followed to extract exact studies and analysis when can help in forming a better review.

The method of unbiased collection approach followed in the studies, and a criterion to judge the quality analysis of these studies as well as the data synthesis and collection procedure followed spanning from a selection of primary studies to the appropriate answers given for the questions all were considered [7]. When an inconsistent set of opinions were discovered, the majority opinions were followed with proper consultation from an expert. Under this section of the study, complete details of the three stages followed were described, which is further related to the step of analyzing the many reports studies in the literature review. As mentioned before, the primary objective of this review is to observe the uses of different search-based methods of SBSE in code modularization.

2.1 Research Questions

The research is based on an objective of investigating the present methods which are followed in the field of SBSM for improving the techniques which are available for making the whole method of refactoring a software more efficient to be used in a real-time setting of software development [3,18,29,39]. The initial seven research questions referred below are equal to the original research study [12,34,50].

The objectives of the research were addresses as per the following set of questions.

Research Question No.	Research Question
Q1.	How can we define Modularization/Re-modularization in SBSE?
Q2.	"How does the proposed multi-objective approach based on NSGA-II perform compared to random search and a mono-objective approach?
Q3.	How does our approach perform compared to existing bugs localization techniques not based on heuristic search?
Q4.	What are the advantages and disadvantages of search-based software engineering techniques and Modularization/Re-modularization?"

Table 1: Research Questions

2.2 The Search Process

Search processes are a manual process that is specific to conference proceedings and research papers since the year 2004 [13,18,19]. The string of search followed was for identifying the set of related studies which were sourced from 5 main databases: IEEEXplore, Information and Software Technology (IST), ACM Digital Library, SpringerLink, ScienceDirect, Empirical Software Engineering Journal, IEEE Proceedings Software, ACM Computer Survey and Wiley Online Library.

Further, Xanthakis et al. [20, 22] make use of search-based methods for the applications of software engineering (testing software), the time of all the studies was selected to be from Jan' 1992 to Dec'2015. ("software" OR "application" OR "project" OR "open source project" OR "product") AND ("Effort" OR "Defect" OR "Fault" OR "Error" OR "Cost" OR "Maintainability" OR "Maintenance Effort" OR "Change" OR "Size") AND ("proneness" OR "prone" OR "prediction" OR "probability" OR "classification" OR "estimation") AND ("Search-based" OR "Evolutionary" OR "Meta-heuristic" OR "heuristic" OR "Multi-objective") AND ("Genetic Programming" OR "Gene Programming" OR "Genetic Algorithm" OR "Particle Swarm Optimization" OR "Ant Colony Optimization" OR "Classifier System" OR "Simulated Annealing" OR

"Tabu Search" OR "Cuckoo Search" OR "Artificial Bee Colony" OR "Evolutionary Programming" OR "Memetic Algorithm" OR "Neuro-evolution" OR "Artificial Immune System (AIS)" OR "Differential Evolution" OR "Harmony Search" OR "Teaching-learning-based Optimization").

These automatic, as well as the efforts, resulted in a set of studies conducted by the authors, and also their references mentioned in the articles were scanned for extracting more and more related studies. However useful insights were also used, along with the list of references, although this data was not included in the whole process of a systematic review and also had no effect on the empirical results. [12, 23-24]

2.3 Inclusion and Exclusion Criteria

Systemic Literature Reviews (SLRs): These are often called literature surveys with formal research questions [5, 16], the search process, data extraction methods, data presentation, and appropriate context with formal keywords.

The excluded research studies are as follows:

- Informal Literature reviews have no research questions, no properly defined search process [11,29], and no proper formal data extraction process.
- Research papers describing the techniques and proceedings used for SBSE and SLRs.

2.4 Data Collection

The data collected from each research study is as follows:

- The source data collected [7,18,19] from the article, journal, or conference proceedings, and the reference details.
- Categorization of the study i.e., type of the research like SLRs, Meta-Analysis [14,19,31], and scope of the research study correlated with technical aspects including functional and nonfunctional domain.
- The author(s) [11,14,29] with their corresponding details including organization and the country.
- Research topic area [23,28,41,50].
- Research issues and questions [22,26].
- Summary of the research study incorporating research issues [12,28,50] and their relevant answers.
- The research study includes SBSE for code modularization papers and its references with the SLRs guidelines [22,41].
- Quality assessment [34] about all the relevant quality attributes of the research study.
- Primary and secondary studies [12,32,49] were used in the research.
- Manual searches on primary and secondary research studies.

Process and Sources

The list of main online sources used to extract the data:

(a) ACM digital library (http://dl.acm.org/)

(b) Google Scholar (http://scholar.google.co.in/)

(c) IEEE Xplore (http://ieeexplore.org/)

(d) Science Direct (http://www.sciencedirect.com/)

(e) Springer LNCS (http://www.springer.com/gp/)

The data collected was analyzed, checked, and extracted. Thus, ensuring quality assessment [22,28,35] of all the data collected from the primary research studies.

2.5 Data Analysis

The data collected was converted into a tabular form to conduct the following:

- The SLRs have referenced SBSE research papers along with the SLR guidelines [22, 27,29,43] which addresses RQ1-RQ3
- The number of literature reviews conducted and published each year [26,29,41,52] with its referenced sources. This also addresses RQ1-RQ3.
- The research topics are covered as per the relevant area of interest by the SLRs [12, 27,31,44,53] with its referenced scope. This addresses RQ4.
- The quality assessment of each SLR has been observed [13] and calculated which addresses RQ4.
- The research trends [16,29,40] and technical area covered addressing RQ6, RQ7.
- The number of primary and secondary studies [1,19,39] conducted in the research study referencing RQ3-RQ4.**2.6 Divergence from the Practice**

While conducting research reviews on different SLRs and articles [13,19,51] few changes were made to the original experimental procedures. These changes are as following:

- The description of major research questions was explained in brief with all the relevant data [15,19].
- SLRs were explained as part of SBSE including code modularization/re-modularization [34,45].
- Different SLRs [12,19,31] were compared in the context of quality parameters such as quality prediction maintainability, defect prediction, effort estimation, and change prediction to attain quality enhancement in terms of code modularization.
- A connection between data collection and data extraction methods [1,23,25,47] with research questions were made to attain the appropriate results.

2.7 Research Question Description

2.7.1 Description of Question 1: How can we define Modularization/Re-modularization in SBSE?

Modularization is usually a software design method that improves the extent to which software consists of distinct interchangeable parts, known as modules [28,29,34].

An important step towards the work will be to count the number of actors associated with the code segment in the respective modules. Modularization of software is a method followed to divide the whole system of software into individual packages (modules) which are expected to loosely and cohesively couple. Although developing software systems to satisfy fresh demands over time, their modularization becomes complicated and their quality gradually loses [26,35].

Unintentionally, the modularization design follows the “divide and conquer” rule in the strategy of solving a problem. This is because numerous benefits are there which could be linked to the modular designing of the software. [34,39]

The benefits of the modularization method are as follows [43,49]:

- It is simpler to keep smaller parts.
- This program could also be categorized as per its functional characteristics.
- The program can include the desired amount of abstraction.
- Components of high-cohesion could also be used again.
- It is feasible to execute simultaneously.
- Security aspects desired.

(a) Modularization-Approach

To simplify the software maintenance operations, appropriate abstractions are developed from the software framework [14,28,34]. The abstractions are recorded variants but are sometimes outdated and no longer used. It points out why the clustering findings of the bunch were not evident at all. Bunch's results were prevalent and structural characteristics independent of whether the MDGs [39,43] represented true systems. Using big clustering outcomes, it is possible to model big landscapes. Using a search-based algorithm, search based clustering algorithms such as Bunch can be assessed. Practitioners need to be reliable in their ability to work perfectly, and this is achieved using systems like a bunch [42,48,50].

There is a thorough analysis of the search space of many open-source systems. Bunch’s clustering effects were illustrated in many ways where the results of individual clustering were also used [27,28,29]. Ducasse S., Alloui I., Abdeen H., Suen M., and Pollet D., [21,27,29] have discussed how the packages are related to one another and also demonstrated these relations. Software that is of large scale is made up of numerous packages. Many developers are unable to grasp how packages are interrelated and positioned. The Blueprint of Package Surface demonstrates a correlation between a package and another and allows it to be easy for the developer to function with or to use them. These packages were represented under the definition of package surfaces [29,31,37]. The category of relationships according to the package they belong to defines the concept of the package surface. The product's structure of inheritance is seen along with the references that are made by a package. [28,35]. Package visualization was conducted successfully where broad applications were used as the inputs. The packages that were not designed well were listed out. Many different software maintainers were used to conduct the tests. [38,41].

(b) Challenges in Optimizing Software Modularization

According to survey challenges in optimizing software, modularization is difficult to handle. Classes in packages are not well spread. Additionally, the coupling of distinct package classes improves the coupling among the packages. Hence, the maximum of the packages depends on some of the dominant ones (that is, the packages containing many classes) [23,26,30].

Optimizing a modularization of quality can generate semantic incoherence: structural and semantic dependencies need to be considered. Most of the current research disregard the effort of refactoring and generate a major bang re-modularization that means that designers alter thousands of lines of code, even for humble applications. It was estimated that designers would have to alter their code by up to 10 percent when adapting alternatives suggested by automated approaches to re-modularization [28,31]. We suggest a multipurpose strategy to resolve these issues to improve the efficiency of software modularization about semantic constraints. The suggested method also seeks to minimize the degree of modification in the solutions generated in terms of achieved improvements. [22,29]

Re-modularization: Large-Scale Refactoring or Re-modularization as defined by Fowler, is another major alteration in the process of implementation and designing, that is limited to the architectural factors. It was done for organizing the entities of architecture in the modules with several interfaces, and to preserve the code behavior.

A new organization could consider various elements of the relationship: prevalent change evaluated coupling of functions. While the significance of re-modularization is quite famous, the method is lengthy in terms of time and a challenging job for staff [29,38,39]. To point out the adjustments to be made, it needs a lot of program understanding.

Re-modularization Analysis using Semantic Clustering: A method of Architecture Recovery proposed originally for extracting the set of the same type of classes in a system is called Semantic Clustering. It works on the lexical similarities present between the vocabularies [16,28]. It further promotes visualization [23,27] of how the package design of the system disperses these sets. To create semantic clusters, four primary features are suggested: weighting and text extraction using the source code, clustering courses with comparable vocabulary, indexing a term using the latent semantic method of indexing, as well as visualizing how suggested clusters were spread across a package system [25,38].

(c) Clustering

The clustering feature operates on a compact matrix where every document was denoted using a vector and the resemblance between the different papers was estimated using the lowest angle cosine created by these vectors [34,37]. A hierarchical agglomerative algorithm for clustering is performed after calculating the resemblance between each pair of papers [45,48]. A single cluster that denotes the domain idea is allocated to each class. The method also produces a tiny set of appropriate terms known as the semantic topics, representing each cluster’s significance (or purpose).

(d) Visualization

Clusters that result from the method of semantic clustering were visualized with the help of Distribution Maps. It is a type of visualization that displays a package in the form of boxes that are filled with squares that represents the class of packages. A class's color reflects the cluster the class belongs to. Two types of data are presented in a Distribution Map: (i) structural data indicates how these classes were structured into sets; (ii) conceptual data refers to semantic clusters distribution as per the colors of classes. [21,25]

2.7.2 Description of Question 2: How does the proposed multi-objective approach based on NSGA-II perform compared to random search and a mono-objective approach?

NSGA-II is an algorithm intended to discover the set of ideal alternatives, known as non-dominated alternatives [23,25,28], including Pareto. This subsection describes a high-level perspective of NSGA-II [22,25] tailored to an issue of re-modularization with the help of structural as well as semantic data. The algorithm requires a code for re-modularization as an input. It begins by generating an individual's random population P0. Then, with the help of alter and selection operators (both mutation and crossover), a Q0 baby population was produced from P0 the parent population [22,25].

Both populations were merged with the R0 the initial population of M size, with a subset of individuals was selected based on dominance principle (the objectives are RRAI, semantic and structural modularization, as well as quality enhancement) and the crowding distance for the same dominance solution for creating next generations [3,12]. The process is repeated with Max iterations. The algorithm's output is the best mixture of refactoring to enhance the quality of software modularization (assessed by structural and semantic metrics) in terms of their effort. [9,11,43]

(a) Model Refactoring using multi-objective optimization

The suggested strategy takes advantage of model refactoring examples and an effective algorithm NSGA-II for suggesting the user automatically refactoring sequences that could be implemented to a specified model. It is the user's responsibility to apply these refactoring algorithms [13,16,32].

For example [13,23,40], this method does not decide on using a new name for a rename refactoring, but it implies applying the operation mentioned for refactoring which helps the user to choose the right name according to their experience.

The figure below introduced an overall framework of this strategy. This required inputs as sed of models that are designed badly, label A [13,54] (that is, current models and its associated refactoring strategies), the models that are designed properly don't require any refactoring (label B), an original model (label C) which further makes use of the software set of metrics (label D) as monitoring parameters [17,28].

The strategy produces a series of refactoring as the output which is used in the original model. The objective of this strategy is for suggesting refactoring that helps in making the design in terms of quality comparable to healthy ones, so we implicitly regarded the reality that refactoring must enhance quality by meeting quality standards comparable to those of excellent design. The method of creating a refactoring sequence (Fig. 1) could be used as a system that discovers a better way of selecting and combining the activities of refactoring out of the ones that are based on the examples in a way as to (a) Maximizing the textual and structural similarities among the entities that are reflected in an original model and are simulated entities of models that are not designed properly, (b) minimizing the structural similarities among the different entities which are refactored in the initial model with the model entities of the models which are well-designed. Structural similarities among these two entities or classes are calculated using these entities' software metrics where their similarities in the texts are estimated using the WordNet-based linear measures. [10,13,15,25]

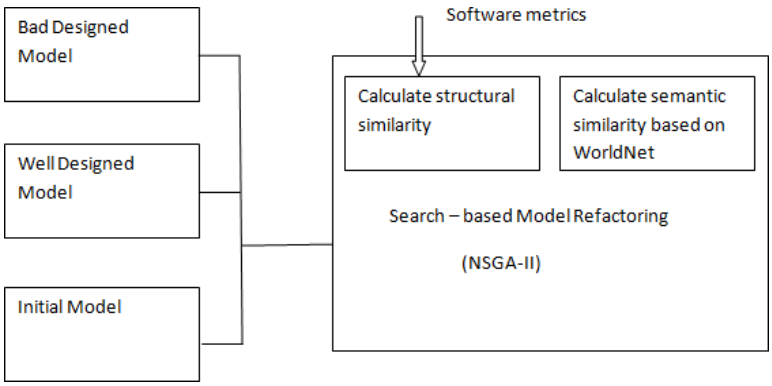

Figure 1. Model refactoring Multi-objective with the help of examples

We compared the performance [25,36] of NSGA-II using a mono-objective and random search genetic algorithm which aggregates every objective into one with an equal measure of weights.

Therefore, when the random search method exceeds the technique of guided search, it can be concluded that formulating this problem is not enough.

It was essential for determining whether the goals were conflicting, carry out a method with the mono objective. The earlier method returns a collection of alternatives that are non-dominating whereas the latter method helps in returning an ideal solution. Hence, a closet solution was chosen to a Knee point [29,33] (that is, a vector consisting of the highest value of objectives among the population members) as the solution of a candidate for comparing with single solutions which are returned by an algorithm with mono-objective [34, 37].

NSGA-II adaptation: We explain how we tailored NSGA-II to discover an optimal trade-ff among the textual and structural resemblance. IAs our goal is to maximize the textual and structural similarities among the given model (the algorithm which is to be implemented) as well as a set of models that are designed badly based on the examples (models undergoing some refactoring) [31,40] & reducing the structural similarities among the model given and a set of models which are designed properly based on examples [33,36,38] (models that do not need any refactoring). In an NSGA-II goal, we separate each criterion. The algorithm requires as its input several examples of model refactoring (our example base), an original model, with a set of metrics. The initial population is built as the individual model sets which represent the solutions possible that are used in the class of an initial model. Here individual means the set of blocks in which every block is comprised of CIM (class

selected from an original model), the CBE (class selected as per the examples) matching to the CIM, and a refactoring list that is a subset of refactoring used in CBE which is used in CIM. [10,13,17,19]

2.7.3 Description of Question 3: How does our approach perform compared to existing bugs localization techniques not based on heuristic search?

(a) Bug Localization: Due to its description, the bug location issue [13,18,20] could be regarded as finding a bug source. To solve the issue, most of the existing researches makes use of the IR (Information-Recovery) method by detecting the semantic and textual similarities between a source code and new report entities. Different IR methods, namely "Latent Semantic Indexing" (LSI), "Latent Dirichlet Allocation" (LDA) [11], and "Vector Space Model" (VSM) [12,16] have been explored. Furthermore, hybrid models were obtained from these IR methods to address the issue of bug location. Based on the above IR methods, we summarize the various instruments and methods suggested in the literature. Bug Scout is a topic-based method that uses LDA for evaluating the data linked to bugs (description, remarks, external links) for detecting the origin of a bug & duplicate bug reports [3,6,10]. However, the precision of Debug Advisor relies on the precision of the description of the report and its precision while describing a bus and its associated code entities. Bug Locator uses the combination of different scores of similarities for bug localization from past reports. It produces a VSM method to be used to remove suspicious source files. Then Bug Locator mines bug reports that were earlier resolved along with associated documents to rank suspect code fragments. [13,18,20]**(b) Fault Localization**

A mix of programming languages such as JavaScript, PHP, and Structure Query Language (SQL) [13,18,32] are published in web applications. Pages are not presented correctly in the web application domain owing to any deformed cross-language mistake like in HTML. These errors can be hard to locate due to the dynamic generation of these codes. Cross-based language fault places are recognized in this article using technique and heuristic search strategies based on traverse assessment.[14] Due to the vibrant generation of these codes, such as HTML [15,21,28]and cross-based language errors can be difficult to find. In the paper, the fault locations of the cross-based language were acknowledged with the help of technical and heuristic search strategies based on traverse evaluation. Depending on these inputs, that causes this application to hurtle, the failures were determined in prior work. It wasn't addressing the issue. Statistical analysis between the failing and passing of a test for discovering the fault location. Additionally, numerous algorithms of localizing the statistical faults were used for finding the whole process of localizing. With the help of Tarantula and Ochiai [15,16,17], the Jaccard algorithm was used to determine the proportion of passed and failed statements testing. For each performed declaration, the suspiciousness rating is calculated to predict the place of the fault.

For each performed declaration, the suspiciousness rating is calculated to predict the place of the fault. In statistical fault localization methods, the enhancements are provided to improve the effectiveness of fault localization.

(c) Heuristic Search

This is a method followed for solving an issue faster than any other conventional method, or in case of finding an exact solution in case conventional method is not able to. This method is used as a shortcut as one of the aspects of velocity accuracy, completeness, optimality, are traded. These search methods are looked at by a heuristic (or heuristic feature) [18,23,28]. It evaluates the data accessible at each branching step and decides on a type of branch that should be followed. This is done when the different alternatives are ranked. A heuristic is defined as a device that is usually very effective but would not offer a high guarantee in all the cases [18,19,25].

2.7.4 Description of Question 4: What are the advantages and disadvantages of search-based software engineering techniques and Modularization/Re-modularization.

Modularization is often used by developers [17,26,34] to simplify their coding. The coding process is broken down into different steps with modularization instead of having to do one big piece of code at a moment.

This technique offers several benefits for designers over other approaches [14,17].

(a) Manageability

One of the benefits of using this approach is that it divides everything into more manageable parts. It could be extremely hard to remain focused on every line of coding when developing a big software program [23,27]. But if a code is broken down among different parts, its analysis doesn't seem to be almost difficult, which helps designers remain [14,17,19] on the job and prevent being overwhelmed by the idea that a specific project has too much to do with it.

(b) Team Programming

As part of the general program, each programmer can be provided an assignment to finish. Then, to generate the program, all the different work [22,26,28] from the programmers is collected at the end. This helps speed up the job and makes it possible to specialize.

(c) Reusable Code

Modular code makes it easy for the developers to use a code again. In case a certain task is sectioned among different classes or functions, it implies that a developed could reuse that specific code when a job is to be performed [10,19]. In case the codes are not organized among different discrete parts, referencing, separating, or implementing that code in other programming situations is more difficult (or impossible) [22,28].

(d) Easier to Debug

This can take a lot of precious time as a programmer searches lines of the code looking for any error that might have occurred and the issues that are created in the program later [14,18]. However, if a program is intended with a modular mindset, every discrete assignment will have its separate code segment.

Hence, if a specific feature has an issue, the programmer understands the source of error and can handle a narrower piece of code [32,35,38].

(e) Reusable Code

In case a certain task is specified to a certain class or function, which enables the developer to reuse a specific code when a job is to be performed. When a code is not categorized among its discrete parts, reference, separation, or implementation of that code in other programming situations is more difficult (or impossible) [29,32].

2.8 Disadvantages of Modularization/Re-modularization

When a programmer wants the code to be as independent or generic as possible when they write modular code, this can make it harder to solve special cases. However, once the data type is no longer a simple integer or float, but an object (which you may not have any idea how to compare because the object will be made far in the future after the sorting function has been coded), things may not be as easy as they now seem [35,37]. The modular method of implementing this type of sorting technique for providing a custom comparison parameter function. Now that implies, that every comparison is a call of function which certainly is a lot heavier than a direct comparison. The drawback is that even integer or float will now suffer the same penalty for performance. Even though the function of comparison is a single relational operator, it is still processed as a function call [37,43,47].

This method was used for dividing a software system among different independent modules and multiple discrete that were

expected to be independently capable of performing tasks. Unintentionally, the design of the model followed the rules of strategies followed for solving problems of “divide and conquer” [30,32,37] as several advantages were also associated with the software’s modular design.

2.9 Advantages and Disadvantages SBSE

A search-based technique has distinctive features, benefits, and disadvantages. Therefore, analyzing the methods closely before using them for a predictive modeling assignment is crucial for a research community as well as software professionals [40,43,49]. However, since only some of the researches address the characteristics of search-based methods, it was hard to simplify the situation that makes it suitable to apply a search-based method [31,34,37].

2.10 Search Algorithms

Many search Algorithms with various names are present which can be used in our strategies. There is a whole family of algorithm which also makes use of some type of characteristics of the search algorithm. According to the structure of the solution which was represented (dependent on a problem), [21,27,43], a number of search operators were applied. Several types of search algorithms were defined which are applied in the entire research. In the remainder of the thesis, an accurate description of the real algorithm used will be provided when a search algorithm is used in a particular issue [45,48].

- **Random Search (RS):** It is the most basic type of search algorithm. It works by sampling the search points randomly and is stopped when an optimum solution was discovered. This technique does not exploit any data about the points that are visited previously while choosing the next points for sampling. Usually, the random search method was used as a base to evaluate the performance of other sophisticated metaheuristics. The most distinguishing property of the RS algorithm is the distribution probability used to sample new solutions. If not stated uniform distribution method was used [26,29].

- **Hill Climbing (HC)-** This method belongs to the local algorithm search class [27,34]. It begins with a search stage and extends to the neighboring alternatives.

A neighboring solution was near structurally, but it depends on the distance concepts between alternatives. In case one neighbor solution holds a better fitness value, the HC “moves” it and looks recursively into a new neighborhood.

If no better neighbor was discovered, HC [32,35] starts with a new solution again. The starting points are often randomly selected. A straightforward approach for visiting the neighborhood could be moving with better fitness to the first neighboring solution discovered. Otherwise, another popular approach would be to assess all of the neighborhood's alternatives and then move to the best [15,18].

- **Alternating Variable Method (AVM):** This is another version of the Hill Climbing method and was used in Korel's early work in software testing [28]. AVM is also a single algorithm like HC which starts its search from a random stage. Then moving to modify the input variables at one time (under software testing). AVM makes use of exploratory search for the chosen variable in which the variables are modified slightly (that is, a neighbor solution similar to HC). In case one neighbor holds a better value of fitness, the exploratory search was listed to be successful. A better neighbor was selected as the new solution [27,29,34].

- **Memetic Algorithms (MAs):** It is a metaheuristic method [29] that utilizes both local and global search (example., an HC GA). Cultural evolution inspires it.

A meme is a cultural transmission imitation unit. The concept is to imitate these memes ' evolution process [30,34]. From the optimization perspective, an MA can be approximately described as a metaheuristic based on population in which, when a local search is used till a local optimum is reached. Using a GA is an easy way to enforce a MA, with the only distinction being that an HC is applied to each generation until a local optimum is achieved.

The price of implementing the local searches was extremely high, hence the size of the population was generally smaller than in GAs and the complete number of generations [35,37,39].

- **Genetic Programming (GP):** [30] is a paradigm for the development of programs for addresses issues such as machine learning [31]. However, the applications of some evolutionary techniques of software production could be traced back to 1985 with Cramer, since Koza [34,37] was commonly known to GP in 1992 with many good applications in real-world issues. Due to the set of input pairs and Y0 the expected output (that is training set: T), the objective of developing a program that is capable of providing the correct answers for every input in T, i.e. g(x)=y0(x, y0)T. [45,47]

The training set, in other words, could also be used as the set of a test case that is to be met. A genetic program was depicted as the tree where every node was a feature with output as child nodes. At each generation, a population of programs is retained, where people are selected to fill the next population according to an issue-specific function of fitness. This function must usually reward minimizing the program mistake when running on the training set [43,47,49].

3. Result Analysis

3.1 Search Results

One of the major parameters is analyzing and assessing the current research work of the researchers. Few of the research articles were excluded [34,36,39] following the criteria of exclusion and inclusion.

The data extracted from each research study has been manipulated in the tabular form and these can be found in Table 3 and Table 4 [15,19,30] which includes the brief detail. Table 3 and Table 4 depicts the research papers that have been published in the last 15 years addressing SBSE for code modularization [13,17,23,30,43].

Name	Approach	Input	Encoding	Mutation	Crossover	Fitness	Outcome	Comments
Antoniol et al. [2003]	Cluster optimization	System containing applications and libraries	Integer array	Two random rows of a column in matrix are swapped or an object is cloned by changing a value from zero to one	A random column is taken as split point and contents are swapped	Inter-library dependencies, number of object-application links and size of libraries	Optimized clustering, sizes and dependencies between libraries diminished	Optimal number of clusters is calculated for a matrix with the Silhouette statistic
Mancoridis et al. [1998]	Automation of partitioning components of a system into clusters	System given as a module dependency graph (MDG)	MDG	N/A	N/A	Minimize inter-connectivity, maximize intra-connectivity, combined as modulariztion quality (MQ)	Optimized clustering of system	
Doval et al. [1999]	Automation of partitioning components of a system into clusters	MDG	String of integers	Standard	Standard	MQ	Optimized clustering of system	Continued work from Mancoridis et al. [1999] by implementing a GA

Table 2: Summary of studies using Search-Based Software method with Clustering

Few research studies have been conducted under peer review research methodology. One of the major parameters is analyzing and assessing the current research work of the researchers [30,35,38].

The data extracted from each research study has been manipulated in the tabular form and these can be found in Table 3 and Table 4 [15,19,32,45] which includes the brief detail.

Name	Approach	Input	Encoding	Mutation	Crossover	Fitness	Outcome	Comments
Di Penta et al. [2005]	A refactoring framework taking into account several aspects of software quality when refactoring existing system.	Software system as a system graph SG	Bit matrix; each library of clusters is represented by a matrix	Swapping two bits in a column or changing a value from 0 to 1 (taking into account preconditions)	N/A	Dependency factor, partitioning ratio, standard deviation and feedback	Refactored libraries	HC and GA used.
Fatiregun et al. [2004]	Program refactoring on source code level	Source code	Integer vector containing transformation numbers	Standard	Standard one-point	Size of source code (LOC)	A sequence of program transformations	Random search, HC and GA are used
Seng et al. [2005]	Optimizing subsystem decomposition for maintenance	Model of system as a graph, extracted from source code	Genes represent subsytem candidates	Split&join, elimination and adoption	Two children from two parents, integrating crossover	Cohesion, coupling, complexity, bottlenecks and cycles	Source code extracted from resulting model	
Seng et al. [2006]	Refactoring a software system with a wide set of operations	Model of system, extracted from source code, with access chains	Ordered list of refactorings	Common class structure refactorings, the list is extended with a suggested transformation	Minimize rejected, duplicated and unused methods and featureless classes and maximize abstract classes	Refactored software system	SA used as search algorithm, introducing a heuristic for weighting conflicting quality goals	
O'Keeffe and Ó Cinnéide [2004]	Automating software refactoring	Software system	N/A	Restructure class hiearchy and method moves, mutations in counter-pairs in order to reverse a move	N/A	Minimize rejected, duplicated and unused methods and featureless classes and maximize abstract classes	Refactored software system	SA used as search algorithm, introducing a heuristic for weighting conflicting quality goals
O'Keeffe and Ó Cinnéide [2006; 2008a]	Automating software refactoring	System as Java source code	N/A	Refactorings regarding visibility, class hierarchy and method placement	N/A	Reusability, flexibility and understandability	Refactored code and design improvement report	Three variations of hill climbing and SA used as search algorithms
O'Keeffe and Ó Cinnéide [2007; 2008b]	Comparison between different search techniques	System as Java source code	Ordered list of refactorings [Seng et al., 2006]	Common class structure refactorings, the list is extended with a suggested transformation [Seng et al., 2006]	A random set of transformations from one parent chosen, the transformations of the other added to that list [Seng et al., 2006]	Understandability	Refactored code and design improvement report	GA and multiple ascent hill climb implemented
Harman and Tratt [2007]	Pareto optimality used for multi-objective optimization	Software system	N/A	Move method	N/A	Coupling and standard deviation of methods per class	A sequence of refactorings	HC used as search algorithm

Table 3: Research Summary of SBSE for Code Modularization

3.2 Algorithms Analyzed

Figure 2 show various kinds of analysis in the algorithm. We have used different algorithms in SBSE, the genetic algorithm 37%, in this algorithm mostly worked are done and they are used mostly. We take different algorithms to check how to work easily and effectively [25,36,54]. The hill-climbing algorithm is also used in this and that prediction is low, many other algorithms are used. In SBSM [45,48,54] and another algorithm is used in 4%, so, the genetic algorithm is used.

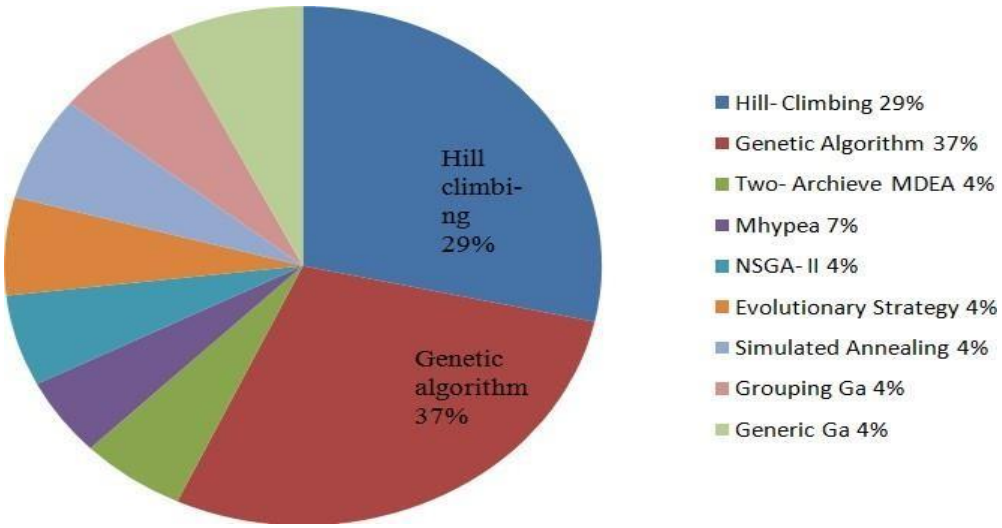

Figure 2. Algorithms Analyzed

4. Conclusion

Our future efforts are to study several other parameters of testing and evaluating them based on these features and arrive at a conclusion that is the parameter that is test schema to enhance the system's effectiveness and stability. The study conducted above includes detailed literature that describes the usage of SBSE since the previous fifteen years. Many issues were listed and discussed in detail as noticed by researchers and practitioners in the many studies they have conducted. This further helps in explaining the data sets which were used in this and their analysis, the methodology they followed as well as the measures of evaluation in the implementation process for solving the problem in focus. This research further offers a brief view of the work which was done already up to date but was not viewed comprehensively for every research conducted in the field. This will help the authors to further explore the issue which is still pending in the field of SBSE and therefore can be resolved by offering an efficient, effective, and viable solution for it.

REFERENCES

[1] International Journal of Computer Applications (0975 – 8887) Innovations in Computing and Information Technology (Cognition 2015) Search-Based Software Engineering Techniques. Arushi Jain M.tech Student ITM University, Gurgaon, Aman Jatain Assistant Professor Amity University, Gurgaon.

[2] Bradbury, J.S., Kelk, D. and Green, M., 2013, May. Effectively using search-based software engineering techniques within model checking and its applications. In Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (pp. 67-70). IEEE Press.

[3] Räihä, O., 2010. A survey on search-based software design. Computer Science Review, 4(4), pp.203-249.

[4] Santos, G., Valente, M.T. and Anquetil, N., 2014, February. Re-modularization analysis using semantic clustering. In 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE) (pp. 224-233). IEEE.

[5] Information & Software Technology, vol. 43, no. 14, pp. 833-839, Dec.2001.

[6] Mahouachi, R., 2018. Search-Based Cost-Effective Software Remodularization. Journal of Computer Science and Technology, 33(6), pp.1320-1336.

[7] B. A. Kitchenham. 2007. Guidelines for performing systematic literature review in software engineering, Technical report EBSE-2007-001, UK.

[8] Rawat, M.S., and Dubey, S.K., 2012. Software defect prediction models for quality improvement: a literature study. International Journal of Computer Science Issues (IJCSI), 9(5), p.288.

[9] Mahouachi, R., 2018. Search-Based Cost-Effective Software Remodularization. Journal of Computer Science and Technology, 33(6), pp.1320-1336.

[10] Ghannem, A., Kessentini, M., Hamdi, M.S., and El Boussaidi, G., 2018. Model refactoring by example: A multi-objective search-based software engineering approach. Journal of Software: Evolution and Process, 30(4), p.e1916.

[11] Dumais, S.T.: 'Latent semantic analysis', Annual review of information science and technology, 2004, 38, (1), pp. 188-23.

[12] Blei, D.M., Ng, A.Y., and Jordan, M.I.: 'Latent Dirichlet allocation', the Journal of Machine Learning Research, 2003, 3, pp. 993-1022.

[13] Zhou, J., Zhang, H., and Lo, D.: 'Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports', in Editor (Ed.) (IEEE, 2012, edn.), pp.14-2.

[14] Artzi, S., Dolby, J., Tip, F. and Pistoia, M., 2011. Fault localization for dynamic web applications. IEEE Transactions on Software Engineering, 38(2), pp.314-335.

[15] J.A. Jones and M.J. Harrold, "Empirical Evaluation of the Tarantula Automatic Fault-Localization Technique," Proc. IEEE/ACM Int'l Conf. Automated Software Eng., pp. 273-282, 2005.

[16] J.A. Jones, M.J. Harrold, and J. Stasko, "Visualization of Test Information to Assist Fault Localization," Proc. Int'l Conf. Software Eng., pp. 467-477, 2002.

[17] R. Abreu, P. Zoeteweij, and A.J.C. van Gemund, "An Evaluation of Similarity Coefficients for Software Fault Localization," Proc. 12th Pacific Rim Int'l Symp. Dependable Computing, pp. 39-46, 2006.

[18] Harman, M., Mansouri, S.A. and Zhang, Y., 2009. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Department of Computer Science, King's College London, Tech. Rep. TR-09-03, p.23.

[19] Harman, M., 2007, June. Search based software engineering for program comprehension. In 15th IEEE International Conference on Program Comprehension (ICPC'07) (pp. 3-13). IEEE.

[20] Rawat, M.S., and Dubey, S.K., 2012. Software defect prediction models for quality improvement: a literature study. International Journal of Computer Science Issues (IJCSI), 9(5), p.288.

[21] Ducasse S, Pollet D, Suen M, Abdeen H, Alloui I (2007) Package surface blueprints: visually supporting the understanding of package relationships. In: Proceedings of an international conference on software maintenance. Paris, France.

[22] Mahouachi, R., 2018. Search-Based Cost-Effective Software Re-modularization. Journal of Computer Science and Technology, 33(6), pp.1320-1336.

[23] Dorigo, M. and Stützle, T., 2003. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In Handbook of metaheuristics (pp. 250-285). Springer, Boston, MA.

[24] Chen, L. and Dey, S., 2001. Software-based self-testing methodology for processor cores. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20(3), pp.369-380.

[25] Mark Harman, "The Current State and Future of Search-Based Software Engineering".

[26] Arcuri, A., 2009. Automatic software generation and improvement through search-based techniques (Doctoral dissertation, University of Birmingham).

[27] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? Journal of Computer and System Sciences, 37(1):79–100, 1988.

[28] B. Korel. Automated software test data generation. IEEE Transactions on Software Engineering, pages 870– 879, 1990.

[29] P. Moscato. On evolution, search, optimization, genetic algorithms, and martial arts: Towards memetic algorithms. Caltech Concurrent Computation Program, C3P Report 826, 1989.

[30]

[31] R. Poli, W. B. Langdon, and N. F. McPhee. A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[32] Mohan, M. and Greer, D., 2018. A survey of search-based refactoring for software maintenance. Journal of Software Engineering Research and Development, 6(1), p.3.

[33] AHUJA, S.P. 2000. A genetic algorithm perspective to distributed systems design. In: Proceedings of the Southeast 2000, 2000, 83 – 90.

[34] AMOUI, M., MIRARAB, S., ANSARI, S. AND LUCAS, C. 2006. A genetic algorithm approach to design evolution using design pattern transformation, International Journal of Information Technology and Intelligent Computing 1 (1, 2), June/ August 2006, 235 – 245.

[35] ANTONIOL, G., DI PENTA, M. AND NETELER, M. 2003. Moving to smaller libraries via clustering and genetic algorithms. In Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR'03), 2003, 307 – 316.

[36] BASS, L., CLEMENTS, P., AND KATZMAN, R. 1998.Software Architecture in Practice, Addison-Wesley, 1998.

[37] BODIN, T., DI PENTA, M., AND TROIANO, L. 2007.A search-based approach for dynamically re-packaging downloadable applications, In Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research (CASCON'07), 2007, 27 – 41.

[38] BOUKTIF, S., KÉGL, B. AND SAHRAOUI, H.2002. Combining software quality predictive models: an evolutionary approach. In: Proceedings of the International Conference on Software Maintenance (ICSM'02) 2002.

[39] BOUKTIF, S., AZAR, D., SAHRAOUI, H., KÉGL, B. AND PRECUP, D. 2004. An improving rule set based software quality prediction: a genetic algorithm-based approach, Journal of Object Technology, 3(4), April 2004, 227 – 241.

[40] BOUKTIF, S. SAHRAOUI, H. AND ANTONIOL, G. 2006. Simulated annealing for improving software quality prediction, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006), 1893 – 1900.

[41] BOWMAN, M., BRIAND, L.C., AND LA BICHE, Y. 2008. Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms, Technical report SCE-07-02, Carleton University.

[42] BRIAND, L., WÜST, J., DALY, J., PORTER, V. 2000. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software, 51, 2000, 245 – 273.

[43] BUI, T. N., AND MOON, B.R. 1996. Genetic algorithm and graph partitioning, IEEE Transactions on Computers, 45(7), July 1996, 841 – 855.

[44] CANFORA, G., DI PENTA, M., ESPOSITO, R., AND VILLANI, M.L. 2005a. An approach for QoS-aware service composition based on genetic algorithms, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) 2005, June 2005, 1069–1075.

[45] CANFORA, G., DI PENTA, M., ESPOSITO, R., AND VILLANI, M.L. 2005b. QoS-aware replanning of composite web services, In Proceedings of IEEE International Conference on Web Services (ICWS'05) 2005, 2005, 121– 129.

[46] CANFORA, G., DI PENTA, M., ESPOSITO, R., AND VILLANI, M.L. 2004. A lightweight approach for QoS-aware service composition. In: Proceedings of the ICSOC 2004 – short papers. IBM Technical Report, New York, USA.

[47] CAO, L., L I, M. AND CAO, J. 2005a. Cost-driven web service selection using genetic algorithm, In LNCS 3828, 2005, 906 – 915

[48] CAO, L., CAO, J., AND LI, M. 2005b. The genetic algorithm utilized in cost-reduction driven web service selection, In LNCS 3802, 2005, 679 – 686.

[49] CHE, Y., WANG, Z., AND LI, X. 2003. Optimization parameter selection using limited execution and genetic algorithms, In APPT 2003, LNCS 2834, 2003, 226–235.

[50] CHIDAMBER, S.R., AND KEMERER, C.F. 1994. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering 20 (6), 1994, 476 – 492.

[51] CLARKE, J., DOLADO, J.J., HARMAN, M., HIERONS, R.M., JONES, B., LUMKIN, M., MITCHELL, B., MANCORIDIS, S., REES, K., ROPER, M., AND SHEPPERD, M. 2003. Reformulating software engineering as a search problem, IEE Proceedings - Software,150 (3), 2003, 161 – 175.

[52] DEB, K. 1999. Evolutionary algorithms for multi-criterion optimization in engineering design, In Proc. Evolutionary Algorithms in Engineering and Computer Science (EUROGEN'99) 135 – 161.

[53] DIPENTA, M., NETELER, M., ANTONIOL, G. AND MERLO, E. 2005. A language-independent software renovation framework, The Journal of Systems and Software 77, 2005, 225 – 240.

[54] DOVAL, D., MANCORIDIS, S., AND MITCHELL, B.S., 1999. Automatic clustering of software systems using a genetic algorithm, In Proceedings of the Software Technology and Engineering Practice, 1999, 73 – 82.

[55] FALKENAUR, E. 1998. Genetic Algorithms and grouping problems, Wiley, 1998

[56] FATIGUE, D., HARMAN, M. AND HIERONS, R. 2004. Evolving transformation sequences using genetic algorithms. In Proceedings of the 4th International Workshop on Source Code Analysis and Manipulation (SCAM 04), Sept. 2004, IEEE Computer Society Press, 65 – 74.

[57] GOLD, N., HARMAN, M., LI AND, Z., AND MAHDAVI, K. 2006. A search-based approach to overlapping concept boundaries. In Proceedings of the 22nd International Conference on Software Maintenance (ICSM 06), USA Sept. 2006, 310 – 319.

[58] GOLDSBY, H. AND CHANG, B.H.C. 2008. Avida-made: a digital evolution approach to generating models of adaptive software behavior. In Proceedings of the Genetic Evolutionary Computation Conference (GECCO 2008), 2008, 1751 – 1758.

[59] [58] GOLDSBY, H., CHANG, B.H.C., MCKINLEY, P.K; KNOESTER, D., AND OFRIA, C.A. 2008. The digital evolution of behavioral models for autonomic systems. In Proceedings of 2008 International Conference on Autonomic Computing, 2008