Harmony Search-Based Approach for Multi-Objective Software Architecture Reconstruction

Amarjeet Prajapati 1,* and Zong Woo Geem 2, *

1 Jaypee Institute of Information Technology, Noida 201309, India
2 College of IT Convergence, Gachon University, Seongnam 13120, Korea
* Correspondence: amarjeetnitkkr@gmail.com (A.P); geem@gachon.ac.kr (Z.W.G)

Received: 19 September 2020; Accepted: 21 October 2020; Published: 31 October 2020

Abstract: The success of any software system highly depends on the quality of architectural design. It has been observed that over time, the quality of software architectural design gets degraded. The software system with poor architecture design is difficult to understand and maintain. To improve the architecture of a software system, multiple design goals or objectives (often conflicting) need to be optimized simultaneously. To address such types of multi-objective optimization problems a variety of metaheuristic-oriented computational intelligence algorithms have been proposed. In existing approaches, harmony search (HS) algorithm has been demonstrated as an effective approach for numerous types of complex optimization problems. Despite the successful application of the HS algorithm on different non-software engineering optimization problems, it gained little attention in the direction of architecture reconstruction problem. In this study, we customize the original HS algorithm and propose a multi-objective harmony search algorithm for software architecture reconstruction (MoHS-SAR). To demonstrate the effectiveness of the MoHS-SAR, it has been tested on seven object-oriented software projects and compared with the existing related multi-objective evolutionary algorithms in terms of different software architecture quality metrics and metaheuristic performance criteria. The experimental results show that the MoHS-SAR performs better compared to the other related multi-objective evolutionary algorithms.

Keywords: software architecture; harmony search; metaheuristic optimization; multi-objective

1. Introduction

The growing demand for various functional and non-functional requirements in modern software systems increases the size and complexity of the implementation code drastically. Maintaining a large and complex software system is a difficult and challenging task for software developers. The architecture of the software is an effective and highly used abstract design model, it enables the developers to deal with the large and complex software systems [1]. The definition of the software architecture is “the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them” [2]. The architecture of the software system can be viewed from the different perspectives of the stakeholder’s concerns. The different architectural views separate the different aspects of the stakeholders [3]. The different architectural views of the software system help in understanding the design decisions and their rationales.

Module view architecture [4] is the most common and important architectural view of a software system where low-level implementation units are organized into higher-level implementation modules/components or sub-systems based on different design criteria. To generate an effective module-view software architecture, multiple design criteria simultaneously (often conflicting) are considered in distributing the low-level implementation units into higher-level modules or sub-systems [5,6]. The definition of low-level implementation units into higher-level modules or
sub-systems may vary according to the software domain and purpose of software architecture reconstruction. In object-oriented software systems, developed in Java programming languages, the classes can be considered as low-level implementation units, and packages can be considered as higher-level modules or sub-systems. So, the package organization of a Java-based object-oriented system can be viewed as module view architecture where architectural module, i.e., packages contains several Java classes.

To produce a qualitative software system, developers usually follow the various architectural design guidelines and principles. The well-designed software architecture makes the system comprehensible, maintainable, testable, and manageable. Despite good quality software system at deployment, over time, the quality of software architecture generally deteriorates [4]. There can be many reasons for the deterioration of software architectural quality. The frequent modifications made over the software system without conforming the architectural guidelines is the key reason. For example, in object-oriented software, the quality of module view software architecture, i.e., quality of package structure deteriorates if the classes are associated or created in the irrelevant packages during the maintenance of the system. It is witnessed that the poorly designed or severely deteriorated software system can be useless if the architecture is not overhauled or reconstructed.

To keep the software systems to be relevant and useful in a frequently changing environment, their architecture often needs to be renovated. The renovation of software architecture is usually done by restructuring the software system based on the different architectural quality criteria [7]. For a large and complex software system, reconstruction of software architecture is a difficult and time-consuming process. The reverse engineering community has developed a variety of approaches to reconstruct the architecture of existing software systems [8]. The infeasibility of manual and deterministic approaches for such systems leads the application of automated search-based metaheuristic approaches. Moreover, the success of search-based software engineering (SBSE) [9] approach opens the various avenue to address the various aspects of software architecture reconstruction problems. Even there has been large progress towards the application of metaheuristic search algorithms in software architecture reconstruction, but still, some effective metaheuristic search algorithms such as harmony search algorithm [10] gained little attention.

Recently, the harmony search algorithm has been successfully tailored and customized by the different research community to solve the various types of optimization problems [11–14]. The flexibility of its customization and effective optimization process makes it a popular metaheuristic algorithm. To overcome shortcomings and various efficiency-related challenges, the operators of the original harmony search algorithm have been customized by the different researchers [15]. Moreover, the simple operators and easy parameter tuning for a diverse set of optimization problems make it more acceptable [15]. The original harmony search algorithm intended to address the single-objective optimization problem. Later many researchers and academics customized the algorithm to address the various types of multi-objective optimization problems [16–19].

Even after the successful customization and application of multi-objective variants of the harmony search algorithm in different areas, it has not been customized and applied to address the software architecture reconstruction. In this work, we customize and apply the harmony search algorithm as a multi-objective harmony search for the software architecture reconstruction problem. The overall framework of the proposed work is named as a multi-objective harmony search for software architecture reconstruction (MoHS-SAR). More specifically, the main contributions of the paper are as follows.

- A framework of multi-objective harmony search for software architecture reconstruction (MoHS-SAR) has been presented. To this contribution, various concepts of multi-objective optimization have been exploited and incorporated into the harmony search algorithm.
- In the proposed MoHS-SAR approach, the external archive concept of TAMoEA, non-dominated sorting and crowding distance concepts of NSGA-II, and candidate solution representation and initialization concepts of MHypEA have been exploited and utilized.
• To demonstrate the effectiveness of the MoHS-SAR, it has been tested on seven object-oriented software projects and compared with the existing related multi-objective evolutionary algorithms. The experimental results show that the MoHS-SAR is competitive in modularization quality (MQ), coupling, cohesion, and inverted generational distance (IGD) with other related multi-objective evolutionary algorithms.

The rest of the article is divided into the following sections: Section 2 presents the related works corresponding to the software architecture reconstruction and multi-objective evolutionary algorithm. Section 3 presents the basic concepts of multi-objective software architecture reconstruction. Section 4 presents a detailed description of the proposed work. Section 5 discusses the experimentation details. Section 6 presents the results and discussions. Section 7 discusses possible threats that may affect the validity of results. Finally, Section 8 concludes with future works.

2. Related Work

In the last two decades, many approaches have been suggested by the software reverse engineering community to reconstruct the architecture of the software system from the existing source code implementation. The existing approaches can be broadly categorized into analytical or deterministic and search-based optimization approaches. The analytical based software architecture recovery approaches (e.g., [20–26]) have been found as a good alternative for small size software applications. However, for the large and complex software applications, the performances of analytical based software architecture recovery approaches degrade drastically. On the other hand, the search-based optimization approaches (e.g., [27–31]) have been found as a good alternative for large and complex software applications. Hence, for the large and complex software systems, the applications of search-based metaheuristics are gaining wide attention among researchers and academics for reconstructing the software systems.

To provide a better understanding on the existing architecture reconstruction approaches, a comprehensive state-of-art survey is presented in the study [8]. The survey is useful for the researchers and academics who want to know the appropriate architecture restructuring methods and would like to reconstruct the architecture from source code information. In another work, the authors [32] provided a detailed process-oriented taxonomy for the software architecture reconstruction. The study [33] introduced a pattern recognition-based software architecture reconstruction approach named as ARM (Architecture Reconstruction Method) where the iterative process used to discover the historical design decision. The approach is based on the deterministic method of software restructuring. The article [34] discussed the Symphony method of software architecture reconstruction, where multiple view-based software architecture approaches are also introduced.

The research described in [35] proposed a plug-in named as Metrics and Architecture Reconstruction Plug-in for Eclipse (MARPLE) for software architecture restructuring and design pattern detection of the source code implementation. The research described in [36] presented a generic metamodel-driven process for reconstructing software architecture and proposed CaCOphoNy. The study [37] discussed the application of software architecture reconstruction to support the renovation of the existing implementation. The authors [38] proposed a software restructuring approach based on graph clustering and partitioning. They reconstructed the software architecture of the existing software systems by improving the cohesion and coupling. The authors [39] used the static and dynamic source code information to reconstruct the static and dynamic view of software architecture. The study [40] proposed a multi-view software architecture reconstruction approach instead of single-view software architecture reconstruction. They especially considered the structure, design, and behavior view of software architecture reconstruction. The study [41] addressed various questions that are relevant to the software architecture reconstruction such as various issues that must be considered, the way of extracting, analyzing, and presenting the information.

The study [42] proposed a software architecture reconstruction approach for Data-Tier software projects. To represent the inter-module dependency for architecture reconstruction hypergraph is used.
The research described in [43] introduced a run-time software architecture reconstruction approach. To recover the run-time software architecture process mining concept was utilized. The research described in [44] presented a behavior-based software architecture approach to reconstruct the up-to-date software architecture. The study [45] conducted a case study to reconstruct the module-view software architecture for the software projects developed in the C# programming language. The research described in [46] proposed an execution-view software architecture reconstruction approach where they used the execution traces, available documentation, and expert knowledge in this purpose. The study [47] provided a systematic analysis and characterization of the proposed/published software architecture reconstruction approaches.

The aforementioned software architecture reconstruction studies have been used successfully to reconstruct the software architecture of various types of software projects. Even these approaches have shown the great results, still, there are some limitations to these approaches. These approaches are mostly based on deterministic/analytical methods in reconstructing software architecture. Hence, for the large and complex software systems, these approaches will take a huge amount of time, and sometimes it can be infeasible to obtain software architecture within time. To overcome this limitation, many researchers and practitioners have suggested the use of metaheuristic search approach to reconstruct the software architecture.

After the incarnation of the Search-Based Software Engineering (SBSE) [9], many software engineering tasks have been formulated as a search-based optimization problem and have been addressed by customizing existing metaheuristic algorithms or by proposing new metaheuristic algorithms. The effectiveness of the SBSE approaches to address the software engineering problems attracted lots of attention to the software engineering community [48]. Recently, the automatic software reconstruction based on the SBSE approach has gained wide attention to the researchers and academics working for reverse engineering field. In the last two decades, many automatic approaches based on SBSE have been proposed to support the software architecture reconstruction.

The authors [28,31] were the first who used the idea of SBSE in restructuring the software system to improve the architectural design. They extracted the software architecture from the implementation using the clustering technique, which is accomplished by the genetic algorithm. The goal of the work was to demonstrate the formulation of software clustering problem as a search-based optimization problem and solving it by customizing the genetic algorithm. Further, the authors [29] developed a tool called Bunch where they implemented the simulated annealing (SA), Hill-climbing (HC), and genetic algorithm (GA). The research described in [30] addressed the software architecture reconstruction problem as a software module clustering problem where the multiple-hill climbing algorithm is used to find out the suitable clustering solution. The research described in [27] also addressed the software architecture reconstruction problem as a software module clustering problem, whereas they used the evolutionary algorithm to obtain the software reconstruction solution. The research described in [28] addressed the software architecture reconstruction problem as a search-based software architecture recovery problem where particle swarm optimization has been used as the search technique. Recently, the research described in [29] addressed the software architecture reconstruction problem as a software remodularization problem where they used the Harmony search algorithm [10] to remodularize the object-oriented software systems.

The search-based software architecture approaches [28–31] discussed in above paragraphs have performed well. The main limitation of these approaches is that they are designed to address the single-objective formulation of software architecture reconstruction. The single-objective search-based software architecture reconstruction approaches are simple but there may be a generation of sub-optimal software reconstruction solution. In other words, searching for the best software architecture solution from the huge search space using the single-objective as a guiding force may lead towards a sub-optimal point of the search-space. Overall, the software remodularization technique in software architecture reconstruction employs a single-objective cannot be an appropriate result for various types of software systems. To overcome the limitations of the single-objective search-based software architecture
reconstruction approaches, researchers and academics have formulated the software architecture reconstruction as a multi-objective search-based software architecture reconstruction problem and solved it using the customized search-based metaheuristic algorithms.

The authors [49] were the first who have introduced the concepts of multi-objective optimization for the software architecture optimization problem. They reformulated the software architecture reconstruction problem as a multi-objective clustering problem and solved it using the multi-objective search algorithm, namely Two-Archive multi-objective evolutionary algorithm (TAMoEA) [50]. The TAMoEA is a genetic evolutionary-based multi-objective optimizer. It exploits the external archive and evolutionary concepts to achieve the good approximation of optimal Pareto front.

The research described in [51,52] customized the non-dominated sorting genetic algorithm-II (NSGA-II) [53] to address the multi-objective software optimization problems related to the module clustering. The NSGA-II is multi-objective extension of canonical genetic algorithm and uses the concept of non-dominated sorting and crowding distance to realize the multi-objective phenomenon. The study [54] introduced a hyper-heuristic-based multi-objective software module clustering approach named as Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA): The MHypEA is demonstrated as an effective approach to address the multi-objective software module clustering problems.

Most of the multi-objective software reconstruction approaches in the literature are based on the genetic algorithm. For example, the TAMoEA, NSGA-II, and MHypEA exploited the crossover and mutation strategies of the standard genetic algorithm to produce new solutions for the next generation. The major limitation in the genetic algorithm-based new solution generation is that the approach involves only two candidate solutions of the population in generation of new solution. However, it has been commonly observed that if all the candidate solutions of the population are involved in generation of new solutions, the final results can be more effective. Keeping this observation, we used the framework of the Harmony Search algorithm in our proposed MoHS-SAR where new candidate solution is generated by considering all the existing candidates of the population and proposed.

Other challenges with the TAMoEA, NSGA-II, and MHypEA is that these approaches provide many options for selection of operators and parameter values and the effectiveness of the final results highly depends on the choices of the operators and the parameter values. So, it is very challenging task to evaluate all operator and parameter combinations for the specific optimization problem. For the software architecture reconstruction problem, which is generally fuzzy in nature, it can be more difficult task to find a most suitable operator and parameter combination. On the other hand, in the proposed MoHS-SAR, there are only few operators and parameter setting values. Hence, without elaborate operator and parameter determination strategy, the proposed MoHS-SAR can generate better results compared to existing multi-objective software reconstruction approaches.

3. Multi-Objective Software Reconstruction Problem

Software architecture reconstruction problem refers to the problem of reconstructing software architecture descriptions from the implementation of the existing system. The architecture reconstruction of a software system is generally required in case of unavailability of documented architecture or availability of poorly documented architecture. The requirements of various views of software architecture documentation needed for the different purposes makes the software architecture reconstruction problem a multifaceted problem. The description of various views of the software architecture is provided in the studies [3,55]. Our software architecture reconstruction problem is mainly focusing on reconstructing module-view of software architecture as our study is concerned with software construction and maintenance.

In module-view software architecture reconstruction, the low-level implementation units are classified into higher-level architectural modules or sub-systems based on some architecture design criteria. In other words, to generate a module-view software architecture more effectively, developers generally follow the various quality criteria simultaneously (often conflicting) in distributing the
The definition of low-level implementation units into higher-level modules or sub-systems [5,6]. The definition of low-level implementation units into higher-level modules or sub-systems may vary according to the software domain and purpose of software architecture reconstruction [56]. In this study, our module-view software architecture reconstruction is representing the package-view software architecture reconstruction of Java-based object-oriented software systems.

Since, in module-view software architecture reconstruction, the software architecture is reconstructed by reorganizing the low-level implementation units into the higher-level architectural modules based on multiple architectural design criteria. Hence, module-view software architecture reconstruction can be seen as a multi-objective optimization problem. In this study, the module-view software architecture reconstruction problem is expressed as an optimization problem with multiple objectives (often conflicting) representing the different architectural design criteria.

The multi-objective optimization problems usually consist of different conflicting objective functions (especially more than one) that requires to be optimized simultaneously. Each objective function of the multi-objective formulation of optimization problem is defined in terms of decision variables. The search space derived from the objective functions and decision variables are usually regarded as objective and decision space. The mathematical model of the multi-objective optimization problem is defined as follows:

\[
\text{Min or Max of } \Phi(x) = [\Phi_1(x), \Phi_2(x), \ldots, \Phi_m(x)]^T \\
\text{Subject to } \psi_i(x) = 0, i = 1, 2, \ldots, p \text{ equality constraints} \\
\chi_j(x) \leq 0, j = 1, 2, \ldots, q \text{ inequality constraints}
\]

Here \( \Phi(x) \) denotes the objective functions and \( x \) represents the decision variable. To define the objective functions for the module-view software architecture reconstruction, multiple architectural design quality metrics representing the needs of the different stakeholders can be considered. Even, there can be a large number of architectural design quality metrics, but in this study, we select a small number of quality metrics. Table 1 provides two sets of multi-objective formulations for module-view software architecture reconstruction. In each set, five objective functions corresponding to the different aspects of software architecture quality are considered. Similarly, there can be many objective functions in multi-objective software architecture reconstruction depending on the quality requirements of stakeholders.

| Architectural Design Quality Metrics-1 (ADQM-1) | Architectural design Quality Metrics-2 (ADQM-2) |
|-----------------------------------------------|-----------------------------------------------|
| Intra-module dependency (maximize)            | Intra-module dependency (maximize)            |
| Inter-module dependency (minimize)            | Inter-module dependency (minimize)            |
| Number of module cycles (minimize)            | Indirect module dependency (minimize)         |
| Implementation units in modules (minimize)     | Implementation units in modules (minimize)     |
| Difference in large and small modules (minimize)| Difference in large and small modules (minimize)|

For module-view software architecture, intra-module dependency (i.e., cohesion) and inter-module dependency (i.e., coupling) are two important architecture design quality metrics. The software system with high intra-module dependency and low inter-module dependency is considered as a good software architecture design. However, along with high intra-module dependency and low inter-module dependency requirements, the development stakeholders’ desires other architectural quality characteristics such as minimum module cycles, all implementation units should not be accommodated in a single module, and there should not be a large difference between large and small modules.

Overall, the software architecture reconstruction problem is formulated as the search-based combinatorial optimization problem where optimal software architecture from feasible software architecture is captured based on multiple objective functions. The literature of software engineering
uses different terms to refer to software architecture reconstruction: software architecture mining, software architecture discovery, software architecture recovery, software reverse architecting, and software architecture extraction.

4. Basics of HS Algorithm

HS [10] is an effective and easily customizable metaheuristic optimization algorithm, especially designed to address different types of search-based optimization problems. The basic idea of the HS is derived from the pitch tuning process of musician to achieve better harmony. HS was originally developed and designed to optimize the problems consisting of only a single objective. However, later it was customized by many researchers and practitioners to address the different types of multi-objective optimization problems. The small number of parameters and easily tailorable to complex optimization problems, makes HS algorithms as a most popular metaheuristic optimization algorithm. Many researchers and academics adapted the HS algorithm to solve the different aspects of complex real-world optimization problems.

To understand the working of the HS algorithm, it is important to know the basic formulation of the optimization problem. The mathematical formulation of the single-objective optimization problem needs to be defined to understand the working the HS algorithm. The definition of a typical single-objective optimization problem in terms of objective function and decision variables can be expressed as follows:

\[
\begin{align*}
\text{Minimize} & \quad f(x) \\
\text{subject to} & \quad x_i \in [L_{B_i}, U_{B_i}], \quad i = 1, 2, \ldots, n
\end{align*}
\]  

where \( f(x) \) is expressed as objective function or goal of the optimization problem and variable \( x = (x_1, x_2, \ldots, x_n) \) is the set of decision variables over which objective function has been defined. Each decision variable can be bounded by upper and lower boundaries. The symbol \( L_{B_i} \) and \( U_{B_i} \) denote lower and upper boundaries of a particular decision variable \( x_i \) respectively.

The HS algorithm begins with the creation of a set of random harmony vector (i.e., candidate solutions) known as population. The randomly generated candidate solutions of the population are improvised as follows: (1) corresponding to each candidate solution of the population, a new candidate solution is created using harmony memory consideration, pitch adjustment, or randomization methods, (2) if the newly generated candidate solution is found better than the worst candidate solution of the population then worst candidate solution is replaced with the newly generated candidate solution otherwise newly generated candidate solution is discarded, (3) Such improvisation is continued until a predefined stopping criterion is met. The detailed description of the HS corresponding to a single-objective problem formulation described in Equation (2) is as follows:

- **Step 1.** Parameters initialization: In HS algorithm, the main parameters are harmony memory size (HMS), harmony memory considering rate (HMCR), and pitch adjusting rate (PAR). The values of these parameters highly affect the performance of algorithm. The settings of these parameters' values may vary from problem to problem. Hence, for a specific optimization problem, there requires proper tuning of these parameter values. Apart from the HMS, HMCR, and PAR, the stopping criterion or the number of improvisations also needed special care while setting its values.
- **Step 2.** Initialize the harmony memory (HM): In harmony memory, candidate solutions are stored as the vectors of the decision variables. In the beginning, the solution vectors are initialized by selecting the random values from the range of each decision variables. For example, the \( j \)th decision variable of \( i \)th solution vector \( x_j^i \) can be initialized as follows:

\[
x_j^i = L_j + \text{rand}() \times (U_j - L_j) \quad \text{where} \quad j = 1, 2, \ldots, N
\]
where $L^j$ and $U^j$ are the lower and upper bound of the $j$th decision variable, $\text{rand}()$ is a random function that generates a random value from a uniform distribution of $[0,1]$.

- **Step 3. Harmony memory improvisation:** In harmony memory improvisation a new solution vector $x_{new} = (x^1_{new}, x^2_{new}, \ldots, x^N_{new})$ i.e., a New Harmony vector is produced using the following three rules: (1) considering harmony memory solution vectors, (2) adjusting the pitch values i.e., decision variable values, and (3) considering the random selection. In harmony memory considering, each decision variable (i.e., $x^j_{new}, x^2_{new}, \ldots, x^N_{new}$) value of a new solution vector $x_{new}$ is selected from the corresponding decision variable values of the harmony memory vectors (i.e., $x^j_{new} \leftarrow \text{rand}(x^j_1, x^j_2, \ldots, x^j_{HMS})$). The consideration of harmony memory vectors decision variable values for the creation of New Harmony vector depends on the harmony memory consideration rate (HMCR), which can be defined as follows:

$$x^j_{new} = \begin{cases} x^j_{new} \in (x^j_1, x^j_2, \ldots, x^j_{HMS}) & \text{if rand} \leq \text{HMCR} \\ x^j_{new} \in (L^j, U^j) & \text{otherwise} \end{cases}$$

(4)

where $U^j - L^j$ contains a possible set of values of $j$th decision variable and $\text{rand}()$ is the uniform random values between 0 and 1. After applying the memory consideration phase, the decision variables of the newly generated solution vector is examined for pitch adjustment. To complete the pitch adjustment operation, the PAR parameter is used as follows:

$$x^j_{new} = \begin{cases} x^j_{new} + \text{rand} \times bw & \text{if rand} \leq \text{PAR} \\ x^j_{new} & \text{otherwise} \end{cases}$$

(5)

where $bw$ is the distance bandwidth parameter. The value of the $bw$ is selected according to the nature of the optimization problems.

- **Step 4. Harmony memory updation:** After generating the new solution vector $x_{new}$ the solution vectors of the harmony vector is updated. If the newly generated solution vector $x_{new}$ is better than the worst solution vector of the harmony memory, the worst solution vector is replaced with the newly generated solution otherwise it is discarded.

- **Step 5. Stopping:** Steps 3 to 4 are repeated until the stopping criterion is satisfied. After the stopping of the algorithm, the best solution of the final harmony memory becomes the final solution.

5. **Proposed Software Architecture Reconstruction Approach**

In this study, we propose a harmony search-based software architecture reconstruction approach named as MoHS-SAR, i.e., multi-objective harmony search for software architecture reconstruction. The general framework of the proposed approach is presented in Figure 1.
The framework of the proposed approach is divided into many different components and each component is dedicated to performing special activities. The working details of each component are provided in the following paragraphs:

- **Extraction of software entities and relationships**: This component is responsible for extraction of various types of source code information required for software architecture reconstruction. For object-oriented software system, source code classes and their dependency relationships are extracted and used for the software architecture reconstruction.

- **Formation of entity dependency graph**: Based on the extracted information of the software classes/entities and their dependency relationships, a class dependency graph (CDG) is constructed. In CDG nodes of the graph represent the classes and the edges represent the class dependency relationships.

- **Representation of software architecture as candidate solution**: To apply the metaheuristic search algorithm, the problems need to be defined and encoded into an appropriate form so that every feasible solution can be generated by applying the search operators. In the context of software architecture reconstruction problem, the intended software system requires to be represented into a suitable form. In this work, we use the integer array encoding method as used by the previous researchers [30,31,57,58].

- **Objective functions**: The main goal of our proposed software architecture reconstruction is to extract the module-view software architecture of the software projects using multi-objective harmony search algorithm. Hence, we need to select the objective functions that can lead the metaheuristic optimization technique in the generation of good approximation of module-view software architecture. In this work, we use the two multi-objective software architecture formulation sets (i.e., ADQM-1, and ADQM-2).

In the software architecture reconstruction, the scales of the defined objective function vary largely. To ensure a reliable distance measure, the objective normalization becomes necessary. Given a software architecture reconstruction solution \(x_1 \in X\), each of its objectives \(f_i(x_1), i = 1, 2, \ldots, M\) is normalized as follows:

\[
f_i^{\text{NORM}}(x_1) = \frac{f_i(x_1) - z^*_i}{z^\text{nad}_i - z^*_i}, i = 1, 2, \ldots, M
\]

where \(z^\text{nad}_i\) and \(z^*_i\) denote the nadir point and ideal point, respectively. Since the Pareto front in case of software architecture reconstruction is unknown, the value of \(z^\text{nad}_i\) and \(z^*_i\) are usually estimated based on the current population.

MoHS-SAR algorithm: To address the multi-objective software architecture reconstruction problem, we design a multi-objective harmony search algorithm by exploiting and integrating the concepts of existing multi-objective optimization approaches. The designed algorithm is referred as multi-objective harmony search for software architecture reconstruction (MoHS-SAR). The pseudocode of the MoHS-SAR is given in Algorithm 1.

The MoHS-SAR contains many supporting concepts that help in performing and controlling the optimization process. To process and store the candidate solutions, three harmony memory is used, i.e., current harmony memory (CHM), new harmony memory (NHM), external harmony memory (EHM). The size of each harmony memory is equal, i.e., HMS. To control the optimization process following parameters are used: number of iteration (NI), harmony memory size (HMS), harmony memory consideration rate (HMCR), external harmony memory consideration rate (EHMCR), and pitch adjustment rate (PAR). The value of these parameter setting has the major influence on the performance of the MoHS-SAR. Hence, appropriate values for these parameters need to be selected. After the selection and initialization of suitable parameter setting of MoHS-SAR, the algorithm proceeds as follows:
Algorithm 1 MoHS-SAR algorithm

1. Set the parameters NI, CHM, NHM, EHM, HMS, HMCR, EHMCR, PAR.
2. Set itr:=0 [iteration initialization]
3. For (i = 1 : i ≤ HMS) do //initialization of CHM
4. Initialize the candidate solutions for the CHM using Equation (7)
5. Compute the objective functions
6. End for
7. Sort the candidate solutions of CHM using non-dominated sorting technique
8. Copy the front 1 candidate solutions into the EHM
9. Repeat
10. Generate candidate solutions for NHM using CHM and EHM
11. For (i = 1 : i ≤ HMS) do
12. For (j = 1 : j ≤ n) do //n is the number of decision variable
13. If (r1 < EHMCR) then //consider the external harmony memory
14. If (r2 < HMCR) then
15. $x_{j_i}^{new} \leftarrow \text{rand}\left(x_{j_1}, x_{j_2}, \ldots, x_{j_{EHMS}}\right)$ //choose a value from EHM
16. If (r3 < PAR) then
17. Perform pitch adjustment
18. End if
19. Else
20. $x_{j_i}^{new} = \text{randInt}(U_j - L_j)$
21. End if
22. Else
23. If (r2<HMCR) then
24. $x_{j_i}^{new} \leftarrow \text{rand}(x_{j_1}, x_{j_2}, \ldots, x_{j_{CHMS}})$ //choose a value from CHM
25. If(r3 < PAR) then
26. Perform pitch adjustment
27. End if
28. Else
29. $x_{j_i}^{new} = \text{randInt}(U_j - L_j)$
30. End if
31. End for
32. End for
33. Sort the candidate solutions of NHM using non-dominated sorting technique
34. Set Q_ad the front 1 of NHM
35. For (i = 1 : i ≤ |Q_ad|) do
36. If (Q_ad[i] cannot be dominated by a solution in EHM) then
37. If (Q_ad[i] dominates any solution in EHM) then
38. Delete dominated solution from EHM
39. Q_ad[i]. flag = 1
40. Else
41. Q_ad[i]. flag = 0
42. End if
43. Else
44. Q_ad[i]. flag = -1
45. End if
46. End for
47. Add the solutions with flag == 1 and flag == 0 to EHM and discard solution with flag == -1
48. If (number of solutions exceeds in EHM to EHM size) then
49. Sort the candidate solutions of EHM using non-dominated sorting
50. Accommodate the solutions of fronts from rank 1 to m
51. If (size of any front > vacant EHM) then
52. Select solutions from front having crowding distance in decreasing order
53. End if
54. End if
55. Set itr = itr+1 //increasing iteration counter
56. Set NHM as CHM
57. Until (itr<NI)//Termination criteria satisfied
58. Produce best non-dominated solution EHM
Step 1: The MoHS-SAR begins with the initialization of CHM. The CHM contains a set of fixed number of candidate solutions. Hence, initialization of CHM requires to initialize each candidate solution residing in CHM. Each decision variable of the candidate solution of CHM is initialized by using the random initialization method defined as follows:

$$x^i_j = \text{randInt}(U^i - L^i) \quad i = 1,2,\ldots,\text{HMS} \quad (7)$$

After initializing candidate solutions of the CHM, the corresponding objective vectors of each candidate solution are computed. For this each objective function value of candidate solution is determined.

Step 2: In this step, the entire candidate solution of CHM is sorted into different non-dominated fronts (i.e., front 1, front 2, ..., front n) using the concept of non-dominated sorting method [24]. The front 1 solutions are superior to rest of the solutions of remaining fronts. To maintain the superior solutions found in every iteration of the algorithm, the solutions of front 1 is transferred into an external harmony memory (EHM) based on EHM updation rule.

Step 3: Generating NHM: It is an important step of the MoHS-SAR, where the candidate solutions of CHM is improvised and stored into NHM. In the harmony memory improvisation process, new candidate solutions for NHM are created, corresponding to each candidate solution of CHM. The decision variable values of each new candidate solution of NHM are determined using the following three rules: 1) Using the candidate solution of CHM or EHM, 2) Adjusting the pitch values i.e., decision variable values, and 3) Considering the random selection.

In CHM or EHM consideration, the decision variable values of new candidate solution are generated based on the decision variable values of candidate solutions resided in the CHM or EHM. The consideration of CHM or EHM depends on the harmony memory consideration rate (HMCR) or external harmony memory consideration rate (EHMCR). After consideration of CHM or EHM, the pitch of each candidate solution is adjusted based on the pitch adjustment rate (PAR). If the CHM or EHM consideration is not applied, then the decision variable value of new candidate solution is generated based on the random selection approach.

Step 4: Updation rule of EHM: In every iteration, the candidate solutions of NHM is sorted based on the non-dominated sorting technique and the candidate solutions of the first front is moved to the EHM based on its updated rule. In this rule, the front 1 candidate solutions are compared with each candidate solution stored in the EHM. If the front 1 candidate solution dominates EHM’s candidate solutions then the dominated solutions of the EHM are removed and front 1 solution is placed in EHM, if the front 1 solution is dominated by all solutions of EHM then it is discarded, if the front 1 solution is non-dominated with all solutions of EHM then it is also inserted into the EHM. If the size of EHM increases by its defined size, then the candidate solutions are sorted using non-dominated sorting techniques and the solutions of front considered for EHM. If any front cannot be accommodated in EHS then crowding distance [24] is used to select partial front for EHM.

Step 5: Stopping: Steps 3 to 4 are repeated until the stopping criterion is satisfied. After the stopping of the algorithm, the best solution of the final EHM becomes the final solution.

6. Experimentation Setup

To carry out a proper investigation about the effectiveness of proposed MoHS-SAR, an appropriate experimentation setup is established. This includes selection of test problems and their representations, configuration of competitive approaches, and suitable parameters settings. The experiments are conducted on a Laptop with Microsoft Windows 10 Home Single Language 64-bit operating system and Intel Core i7-1160G7 4.4 GHz CPU. Since the proposed and existing approaches are stochastic optimizers, they may not produce the same output on the different run for the same problem instance. Hence, we run each algorithm 31 times and applied the Wilcoxon rank-sum test to evaluate the quality measures of software architecture reconstruction solutions.
6.1. Test Problems

To demonstrate the effectiveness of the proposed MoHS-SAR, seven object-oriented software projects especially developed in the Java programming languages have been selected for experimentation. These software projects are open-source and easily available. To avoid the biased implication regarding the results, the software projects having different size and complexity level are included. A brief description of the selected software systems is presented in Table 2.

| Software Systems | #Dependencies | #Classes |
|------------------|---------------|---------|
| JFreeChart-0.9.21| 1420          | 401     |
| JHotDraw-6.0b1   | 2125          | 398     |
| JavaCC-1.5      | 722           | 154     |
| JUnit-3.81      | 276           | 100     |
| Java Servlet API-2.3 | 131     | 63     |
| XML API DOM-1.0. b2 | 209     | 119     |
| DOM 4J-1.5.2    | 930           | 195     |

The main reason for inclusion of these software projects in our experimentation is that these software projects have diverse characteristics and have been used by many researchers and academics for different software engineering research purposes [57–60]. To extract the various required attributes such as classes and their relationships of the software systems, we followed the same tools and methodology used in the works [57–60].

6.2. Competitive Approaches

In the literature of SBSE, many multi-objective optimization algorithms dedicated to addressing the different aspects of software engineering problems have been proposed. To justify the effectiveness of the proposed MoHS-SAR, we have selected some related multi-objective optimization algorithms addressing the similar software engineering problems for the comparison. A brief description of these selected metaheuristic optimization algorithms is given below.

- **Two-Archive multi-objective evolutionary algorithm (TAMoEA):** The TAMoEA [50] was designed to address the multi-objective software module clustering problem. It is a genetic evolutionary-based multi-objective optimizer. It exploits the external archive and evolutionary concepts to achieve the good approximation of optimal pareto front.

- **Non-dominated sorting genetic algorithm-II (NSGA-II):** The NSGA-II [53] is multi-objective extension of canonical genetic algorithm and uses the concept of non-dominated sorting and crowding distance to realize the multi-objective phenomenon. This algorithm has also been tailored to address the multi-objective software module clustering [60].

- **Multi-objective Hyper-heuristic Evolutionary Algorithm (MHypEA):** The MHypEA [53] is especially designed to solve the multi-objective software module clustering. This algorithm exploits the concepts of hyper-heuristics and evolutionary optimization to achieve the multi-objective optimization behaviour.

The main reason for selection of the above algorithms is that these algorithms were initially designed or customized to address the multi-objective software module clustering problem. But they can be easily used to address the multi-objective software architecture reconstruction problems as the software module clustering technique can be used to reconstruct the module-view of software architecture.
6.3. Parameter Settings

The parameter settings of the search-based metaheuristic algorithms play an important role in the performance of the results. [61, 62] To make an appropriate parameter value, we use the parameter values as suggested and used by the original manuscript of the competitive algorithms (i.e., TAMoEA, NSGA-II, and MHypEA). The details of the parameter settings of each algorithm are given in Table 3.

| Parameters        | TAMoEA | NSGA-II | MHypEA | MoHS-SAR |
|-------------------|--------|---------|--------|----------|
| Population size   | 10N    | 10N     | 10N    | 10N      |
| Crossover         | 0.8    | 0.8     | 0.8    | —        |
| Mutation          | 0.04log₂N | 0.04log₂N | 0.04log₂N | —        |
| HMCR              | —      | —       | —      | 0.65     |
| EHMCR             | —      | —       | —      | 0.65     |
| PAR               | —      | —       | —      | 0.35     |
| Bw                | —      | —       | —      | 0.45     |
| Archive size      | 10N    | —       | —      | 10N      |
| Number of fitness evaluations | 2000*N | 2000*N | 2000*N | 2000*N |

*N is the number of software entities.

As the size of software systems (i.e., number of entities) used for the validation of the proposed approach is not the same, hence using the same size for the population and other parameters is not appropriate. Therefore, parameter values of the algorithms have been defined in terms of the number of entities present in the particular software system.

7. Results and Discussion

This section is intended to show the effectiveness of the proposed MoHS-SAR presented in this study to address the problem of software architecture reconstruction. This section is divided into four parts, as follows: The first part is designed to study the improvement of MQ software architecture quality. Second part is related to the software coupling measurement. Third part is concerned with the cohesion measurement. The last part is dedicated to IGD measurement. Each of these parts is associated with both ADQM-1 and ADQM-2 multi-objective software architecture reconstruction formulation and TAMoEA, NSGA-II, MHypEA, and MoHS-SAR algorithms.

Since the proposed and existing approaches are stochastic optimizer, hence, we used the Wilcoxon rank sum test to evaluate the quality measures of software architecture reconstruction solutions. In this study, the significance level for the result evaluation is set to 0.05. In every part of the result demonstration, the performance of proposed MoHS-SAR is also compared with the performance of TAMoEA, NSGA-II, and MHypEA algorithms. The comparative results are signified with symbols of “[−]”, “[+]”, and “[≈]”. The symbols “[−]”, “[+]”, and “[≈]” used in the Tables 4–7 denote that the relative performance of the TAMoEA, NSGA-II, and MHypEA algorithms is significantly worse than, better than, and not significantly different than that of the proposed MoHS-SAR, respectively.
Overall, the proposed and existing approaches are evaluated with ADQM-1 and ADQM-2 multi-objective formulation designed for the software architecture reconstruction problem. First, the comparison of MoHS-SAR with TAMEA, NSGA-II, and MHypEA in terms of software quality metrics (i.e., MQ, coupling, and cohesion) with ADQM-1 and ADQM-2 multi-objective formulation is described. Secondly, the comparison of MoHS-SAR with TAMEA, NSGA-II, and MHypEA in terms of Pareto quality indicator metrics (i.e., IGD) with ADQM-1 and ADQM-2 is presented. Finally,
a summarized analysis of the software quality metrics (i.e., MQ, coupling, and cohesion) with ADQM-1 and ADQM-2 is provided.

Table 4 presents the MQ results obtained through proposed MoHS-SAR and existing approaches TAMoEA, NSGA-II, and MHypEA with all seven subject systems under both ADQM-2 and ADQM-1 multi-objective formulation. In ADQM-2 multi-objective formulation, the proposed MoHS-SAR performing significantly better to the existing approaches TAMoEA, NSGA-II, and MHypEA in most of the cases. The proposed MoHS-SAR is significantly better in four cases, five cases, and four cases to the MHypEA, TAMoEA, and NSGA-II, respectively. Similar to the ADQM-2 multi-objective formulation, the proposed MoHS-SAR is performing significantly better to the existing approaches TAMoEA, NSGA-II, and MHypEA in most of the cases.

Table 5 presents coupling as an assessment criterion to evaluate the performance of proposed MoHS-SAR and existing approaches TAMoEA, NSGA-II, and MHypEA. The lower the value of coupling is considered as better software architecture reconstruction solution. If we see the coupling results provided in Table 5, it is clear that the proposed MoHS-SAR approach has the lower coupling values compared to the existing approaches TAMoEA, NSGA-II, and MHypEA in most of the cases and these values are significantly lower in most of the cases. This is true for both the multi-objective formulations ADQM-1 and ADQM-2.

The cohesion values of software architecture reconstruction achieved with the proposed MoHS-SAR and existing approaches TAMoEA, NSGA-II, and MHypEA is presented in Table 6. These cohesion results are collected under both ADQM-2 and ADQN-1 multi-objective software architecture reconstruction formulations. In contrast to the coupling values of software architecture reconstruction solution, the higher values of the cohesion of software architecture reconstruction solution is considered as good quality software architecture. The cohesion results presented in Table 6 demonstrate that the proposed MoHS-SAR is able to generate the software architecture reconstruction solutions with higher cohesion values compared to the existing approaches TAMoEA, NSGA-II, and MHypEA. The cohesion results show that the proposed MoHS-SAR is producing significantly larger cohesion values to the existing approaches TAMoEA, NSGA-II, and MHypEA in most of the cases. If we see the cohesion results of both ADQM-2 and ADQN-1 multi-objective software architecture reconstruction, the proposed MoHS-SAR is producing significantly larger cohesion values to the existing approaches TAMoEA, NSGA-II, and MHypEA in most of the cases.

The MQ, coupling, and cohesion assessment criteria are used to determine the quality of software architecture design. The MQ, coupling, and cohesion results reported in Tables 4–6, showed that the proposed MoHS-SAR approach outperforms the existing TAMoEA, NSGA-II, and MHypEA approaches. In other words, the better MQ, coupling, and cohesion value of the proposed MoHS-SAR compared to the TAMoEA, NSGA-II, and MHypEA indicates that the proposed approach is able to reconstruct better software architecture compared to the existing TAMoEA, NSGA-II, and MHypEA approaches. Hence, the proposed MoHS-SAR approach can be considered as a good alternative for software architecture reconstruction.

Even if the proposed MoHS-SAR approach is performing better in terms of the software quality metrics to the TAMoEA, NSGA-II, and MHypEA approaches, its performance needs to be validated with Pareto quality indicators to justify its significance from the algorithmic perspective. For this, we used the IGD quality indicator to evaluate the performance of multi-objective optimization algorithms. In Table 7, the results of IGD Pareto quality indicator obtained through the proposed MoHS-SAR approach and existing TAMoEA, NSGA-II, MHypEA approaches with ADQM-2 and ADQM-1 multi-objective formulation are presented. The IGD results clearly indicate that the proposed MoHS-SAR is performing better compared to the existing TAMoEA, NSGA-II, MHypEA approaches in most of the cases.

Figure 2 presents a summarized view of the MQ, coupling, and cohesion results obtained through the proposed MoHS-SAR approach and the existing approaches TAMoEA, NSGA-II, and MHypEA under both ADQM-1 and ADQM-2 multi-objective software architecture reconstruction formulations.
From Figure 2, it is clear that the proposed MoHS-SAR approach can generate software architecture solution having better MQ, coupling, and cohesion values compared to the existing TAMoEA, NSGA-II, and MHypEA approaches.

Overall, the proposed MoHS-SAR approach produces better results compared to the existing multi-objective software architecture reconstruction approaches. There can be multiple reasons for producing such good results. The existing approaches TAMoEA, NSGA-II, and MHypEA are based on the concept of the evolutionary approach, i.e., Genetic Algorithm where the off-spring is created using the two-parent vectors. On the other hand, in MoHS-SAR, new candidate vector is generated by considering all the existing candidate vectors. This feature of the MoHS-SAR makes the approach flexible and effective in the generation of better results. The next reason is that the MoHS-SAR requires only a few numbers of parameters to be determined before the run of the algorithm. Whereas, the TAMoEA, NSGA-II, and MHypEA approaches provide many options for operators and parameters and the effectiveness of the approaches highly depends on the choices of the operators and the parameters’ values. It is very tedious to evaluate all operator and parameter combinations for the specific optimization problem. Without an elaborate operator and parameter determination strategy, the proposed MoHS-SAR can generate better results compared to the TAMoEA, NSGA-II, and MHypEA approaches.

Another reason for such good results of the proposed MoHS-SAR approach could be the effective maintenance of best non-dominated solutions obtained in an individual iteration of the algorithm. The proper customization of metaheuristic algorithms suit the optimization problem always helps in producing good quality solutions. The hybridization of multiple concepts of different metaheuristic algorithms also leads to the generation of good quality solution. In the proposed MoHS-SAR approach, the external archive concept of TAMoEA, non-dominated sorting and crowding distance concepts of
NSGA-II, and candidate solution representation and initialization concepts of MHypEA have been exploited and utilized.

Even our proposed MoHS-SAR approach performs well compared to the existing multi-objective software reconstruction approaches in terms of MQ, coupling, cohesion, and IGD, still there are many limitations and differences compared to the traditional software architecture recovery approaches. Our proposed approach mainly focuses on structural module view of software architecture reconstruction and is not guaranteed to generate architectural solution compliance with the developers’ view of software architecture. Many traditional approaches are relied on the textual or non-structural input (e.g., identifiers names, method names and file authorship) to generate the architectural components. The design of software architecture reconstruction approaches also varies according to the programming languages used for the software systems. Our proposed approach is designed for the software systems developed in the object-oriented programming languages. But our approach can also work with the software systems developed in other programming languages by redefining the meaning of software entity and component. In summary, the fuzzy nature and multiple views of the software architecture encourage the designing of different types of software architecture reconstruction approaches.

8. Threats to Validity

The proposed approach is based on the concepts of the search-based metaheuristic optimization where various operators and parameters are used to make an effective search. The better coordination of operators and well-suitable values of the parameters can lead the search-based metaheuristics towards more appropriate optimization results. Apart from the methodology, the modeling of the problem can also affect the outcome of the approach. In this section, we discuss various types of factors that can affect the validity of the outcome obtained through the methodology. More specifically, we have considered the external and internal validity.

In external threats to validity, those factors are considered that could affect the results of the approach for the broad perspective of the problems. In software architecture reconstruction, the software system to be reconstructed are generally contains different characteristics. Some software system may be small and less complex, and some software system may be large and high complexity. Hence, generalizing the results to broader categories of software through some specific software is very challenging. Some approaches may perform better over some set of software systems; however, some approaches may perform better over some other set software systems. This threat to validity for our approach has been mitigated by transforming the software systems into an abstract representation called entity dependency graph. Since, in this abstraction, many of the software systems can be transformed into a certain set of entity dependency graphs. Further, to validate the approach, software systems with various size and complexity have been considered in our experimentation.

In internal threats to validity, the degree of relationships of the independent variable to the dependent is considered. In other words, the internal threat to validity corresponds to the degree of cause-and-effect relationships to which conclusion can be drawn about the causal effect of independent variables on the dependent variables. In our approach, possible internal threats to validity can be selection and assumption about the statistical analysis.

9. Conclusions and Future Works

This paper has proposed an HS-based multi-objective optimization algorithm for solving software architecture reconstruction problem, called MoHS-SAR. In the proposed MoHS-SAR, the concepts of external harmony memory and crowding distance have been exploited effectively and incorporated in the standard harmony search framework. The experimental results have demonstrated that the MoHS-SAR can produce better software architecture reconstruction problems than several existing customized multi-objective software architecture recovery approaches, such as TAMoEA, NSGA-II, and MHypEA. According to the experimental results, the proposed approach can generate software
architecture reconstruction solution having better values of MQ, coupling, cohesion, and IGD quality metrics. However, the proposed approach does not guarantee to generate a good architectural solution from the perspective of other architectural views than the structural module view of software architecture. The proposed approach works well with a limited number of objectives, but for the software architecture problems having a large number of objectives may not perform well. So the proposed approach needs to be further improved for many-objective software architecture reconstruction problems. Future works include the consideration of multiple dimensions of source code information (e.g., lexical, dynamic, etc) along with more quality aspects as the objective function in software architecture reconstruction. To strengthen the supremacy of the proposed approach, it needs to experiment over more complex and large software systems.

Author Contributions: Conceptualization, A.P.; Methodology, A.P.; Implementation, A.P.; Writing—Original Draft Preparation, A.P.; Supervision, Z.W.G.; Writing—Review and Editing, A.P., Z.W.G.; Funding, Z.W.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2020R1A2C1A01011131).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kouroshfar, E.; Mirakhorli, M.; Bagheri, H.; Xiao, L.; Malek, S.; Cai, Y. A Study on the Role of Software Architecture in the Evolution and Quality of Software. In Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories, Florence, Italy, 16–17 May 2015; pp. 246–257.
2. Bass, L.; Clements, P.; Kazman, R. Software Architecture in Practice; Addison-Wesley: Boston, MA, USA, 1998.
3. Kruchten, P. The 4+1 view model of architecture. IEEE Softw. 1995, 12, 42–50. [CrossRef]
4. Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; Stafford, J. Documenting Software Architectures: Views and Beyond; Pearson Education: London, UK, 2010.
5. Garcia, J.; Popescu, D.; Edwards, G.; Medvidovic, N. Identifying architectural bad smells. In Proceedings of the 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany, 24–27 March 2009; pp. 255–258.
6. Hochstein, L.; Lindvall, M. Combating architectural degeneration: A survey. Inf. Softw. Technol. 2005, 47, 643–656. [CrossRef]
7. Mirakhorli, M. Software architecture reconstruction: Why? What? How? In Proceedings of the IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Montreal, QC, Canada, 2–6 March 2015; p. 595.
8. Pollet, D.; Ducasse, S.; Poyet, L.; Alloui, I.; Campan, S.; Verjus, H. Towards a process-oriented software architecture reconstruction taxonomy. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR’07), Amsterdam, The Netherlands, 21–23 March 2007; pp. 137–148.
9. Harman, M.; Jones, B.F. Search-based software engineering. Inf. Softw. Technol. 2001, 43, 833–839. [CrossRef]
10. Geem, Z.W.; Kim, J.H.; Loganathan, G. A new heuristic optimization algorithm: Harmony search. Simulation 2001, 76, 60–68. [CrossRef]
11. Toklu, Y.C.; Bekdaş, G.; Geem, Z.W. Harmony Search Optimization of Nozzle Movement for Additive Manufacturing of Concrete Structures and Concrete Elements. Appl. Sci. 2020, 10, 4413. [CrossRef]
12. Nasir, M.; Sadollah, A.; Yoon, J.H.; Geem, Z.W. Comparative Study of Harmony Search Algorithm and its Applications in China, Japan and Korea. Appl. Sci. 2020, 10, 3970. [CrossRef]
13. Bekdaş, G.; Arama, Z.A.; Kayabekir, A.E.; Geem, Z.W. Optimal Design of Cantilever Soldier Pile Retaining Walls Embedded in Frictional Soils with Harmony Search Algorithm. Appl. Sci. 2020, 10, 3232. [CrossRef]
14. Saha, S.; Ghosh, M.; Ghosh, S.; Sen, S.; Singh, P.K.; Geem, Z.W.; Sarkar, R. Feature Selection for Facial Emotion Recognition Using Cosine Similarity-Based Harmony Search Algorithm. Appl. Sci. 2020, 10, 2816. [CrossRef]
15. Zhang, T.; Geem, Z.W. Review of harmony search with respect to algorithm structure. Swarm Evol. Comput. 2019, 48, 31–43. [CrossRef]
16. Sivasubramani, S.; Swarup, K.S. Multi-objective harmony search algorithm for optimal power flow problem. Int. J. Electr. Power Energy Syst. 2011, 33, 745–752. [CrossRef]
17. Sivasubramani, S.; Swarup, K.S. Environmental/economic dispatch using multi-objective harmony search algorithm. *Elec. Power Syst. Res.* **2011**, *81*, 1778–1785. [CrossRef]

18. Nekooei, K.; Farsangi, M.M.; Nezamabadi-Pour, H.; Lee, K.Y. An Improved Multi-Objective Harmony Search for Optimal Placement of DGs in Distribution Systems. *IEEE Trans. Smart Grid* **2013**, *4*, 557–567. [CrossRef]

19. Qu, B.Y.; Li, G.S.; Guo, Q.Q.; Yan, L.; Chai, X.Z.; Guo, Z.Q. A Niching Multi-Objective Harmony Search Algorithm for Multimodal Multi-objective Problems. In *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand, 10–13 June 2019; pp. 1267–1274.

20. Lung, C.H. Software architecture recovery and restructuring through clustering techniques. In *Proceedings of the Third International Workshop on Software Architecture (ISAW ’98)*, New York, NY, USA, November 1998; Association for Computing Machinery: New York, NY, USA, 1998; pp. 101–104.

21. Eixelsberger, W.; Ogris, M.; Gall, H.; Bellay, B. Software architecture recovery of a program family. In *Proceedings of the 20th International Conference on Software Engineering*, Kyoto, Japan, 19–25 April 1998; pp. 508–511.

22. Sartipi, K. Alborz: A query-based tool for software architecture recovery. In *Proceedings of the 9th International Workshop on Program Comprehension. IWPC 2001*, Toronto, ON, Canada, 12–13 May 2001; pp. 115–116.

23. Sartipi, K. Software architecture recovery based on pattern matching. In *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, Amsterdam, The Netherlands, 22–26 September 2003; pp. 293–296.

24. Pashov, I.; Riebisch, M. Using feature modeling for program comprehension and software architecture recovery. In *Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, Brno, Czech Republic, 27–27 May 2004; pp. 406–417.

25. Zhang, Q.; Qiu, D.; Tian, Q.; Sun, L. Object-oriented software architecture recovery using a new hybrid clustering algorithm. In *Proceedings of the 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery*, Yantai, China, 10–12 August 2010; pp. 2546–2550.

26. Fontana, A.; Roveda, R.; Zanoni, M.; Raibulet, C.; Capilla, R. An Experience Report on Detecting and Repairing Software Architecture Erosion. In *Proceedings of the 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Venice, Italy, 5–8 April 2016; pp. 21–30.

27. Doval, D.; Mancoridis, S.; Mitchell, B. Automatic Clustering of Software Systems Using a Genetic Algorithm. In *Proceedings of the Ninth International Workshop Software Technology and Engineering Practice*, Pittsburgh, PA, USA, 2 September 1998; pp. 73–81.

28. Mancoridis, S.; Mitchell, B.S.; Rorres, C.; Chen, Y.F.; Gansner, E.R. Using automatic clustering to produce high-level system organizations of source code. In *Proceedings of the International Workshop Program Comprehension*, Ischia, Italy, 24–26 June 1998; pp. 45–53.

29. Mancoridis, S.; Mitchell, B.S.; Chen, Y.; Gansner, E.R. Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *Proceedings of the IEEE International Conference on Software Maintenance*, Oxford, UK, 30 August–3 September 1999; pp. 50–59.

30. Mahdavi, K.; Harman, M.; Hierons, R.M. An multiple hill climbing approach to software module clustering. In *Proceedings of the International Conference on Software Maintenance*, Amsterdam, The Netherlands, 22–26 September 2003; pp. 315–324.

31. Praditwong, K. Solving software module clustering problem by evolutionary algorithms. In *Proceedings of the 2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Nakhon Pathom, Thailand, 11–13 May 2011; pp. 154–159.

32. Ducasse, S.; Pollet, D. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE Trans. Softw. Eng.* **2009**, *35*, 573–591. [CrossRef]

33. Guo, G.Y.; Atlee, J.M.; Kazman, R. A Software Architecture Reconstruction Method. In *Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, TX, USA, 22–24 February 1999.

34. Deursen, A.V.; Hofmeister, C.; Koschke, R.; Moonen, L.; Riva, C. Symphonic: View-driven software architecture reconstruction. In *Proceedings of the Third Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, Oslo, Norway, 12–15 June 2004; pp. 122–132.

35. Fontana, F.A.; Zanoni, M. A tool for design pattern detection and software architecture reconstruction. *Inf. Sci.* **2011**, *181*, 1306–1324. [CrossRef]
36. Favre, J. CaCOphoNy: Metamodel-driven software architecture reconstruction. In Proceedings of the 11th Working Conference on Reverse Engineering, Delft, The Netherlands, 8–12 November 2004; pp. 204–213.

37. O’Brien, L.; Smith, D.; Lewis, G. Supporting Migration to Services using Software Architecture Reconstruction. In Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP’05), Budapest, Hungary, 24–25 September 2005; pp. 81–91.

38. Șora, I.; Glodean, G.; Gligor, M. Software architecture reconstruction: An approach based on combining graph clustering and partitioning. In Proceedings of the 2010 International Joint Conference on Computational Cybernetics and Technical Informatics, Timisoara, Romania, 27–29 May 2010; pp. 259–264.

39. Riva, C.; Rodriguez, J.V. Combining static and dynamic views for architecture reconstruction. In Proceedings of the Sixth European Conference on Software Maintenance and Reengineering, Budapest, Hungary, 11–13 March 2002; pp. 47–55.

40. Sartipi, K.; Dezhkam, N.; Safyallah, H. An Orchestrated Multi-View Software Architecture Reconstruction Environment. In Proceedings of the 2006 13th Working Conference on Reverse Engineering, Benevento, Italy, 23–27 October 2006; pp. 61–70.

41. Deursen, A.V.; Riva, C. Software architecture reconstruction. In Proceedings of the 26th International Conference on Software Engineering, Edinburgh, UK, 28–28 May 2004; pp. 745–746.

42. Ersoy, E.; Kaya, K.; Altunışık, M.; Sözer, H. Using Hypergraph Clustering for Software Architecture Reconstruction of Data-Tier Software. In Software Architecture. ECSA 2016; Tekinerdogan, B., Zdun, U., Babar, A., Eds.; Springer: Cham, Switzerland, 2016; pp. 326–333.

43. Jong, T.D.; Werf, J.M.E.M.V.D. Process-mining based dynamic software architecture reconstruction. In Proceedings of the 13th European Conference on Software Architecture (ECSA ’19), Paris, France, 9–13 September 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 217–224.

44. Nicolaescu, A.; Lichter, H. Behavior-Based Architecture Reconstruction and Conformance Checking. In Proceedings of the 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, Italy, 5–8 April 2016; pp. 152–157.

45. Pruijt, L.; Wiersema, W. Software architecture reconstruction and compliance checking: A case study. In Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings (ECSA’17), Canterbury, UK, 11–15 September 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 145–151.

46. Ahn, H.; Kang, S.; Lee, S. Reconstruction of execution architecture view using dependency relationships and execution traces. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC ’18), Pau, France, 9–13 April 2018; Association for Computing Machinery: New York, NY, USA; pp. 1417–1424.

47. Uzun, B.; Tekinerdogan, B. Chapter 4—Domain-driven analysis of architecture reconstruction methods. In Model Management and Analytics for Large Scale Systems; Tekinerdogan, B., Babur, Ö., Cleophas, L., van den Brand, M., Aksamit, M., Eds.; Academic Press: Cambridge, MA, USA, 2020; pp. 67–84.

48. Hussain, I.; Khanum, A.; Abbasi, A.Q.; Javed, M.Y. A Novel Approach for Software Architecture Recovery Using Particle Swarm Optimization. Int. Arab J. Inf. Technol. 2015, 12, 32–41.

49. Praditwong, K.; Harman, M.; Yao, X. Software module clustering as a multi-objective search problem. IEEE Trans. Softw. Eng. 2011, 37, 264–282. [CrossRef]

50. Praditwong, K.; Yao, X. A New Multi-Objective Evolutionary Optimisation Algorithm: The Two-Archive Algorithm. In Proceedings of the 2006 International Conference on Computational Intelligence and Security, Guangzhou, China, 3–6 November 2006; pp. 286–291.

51. Barros, M. An analysis of the effects of composite objectives in multiobjective software module clustering. In Proceedings of the Fourteenth International Conference on Genetic and Evolutionary, Philadelphia, PA, USA, 7–11 July 2012; pp. 1205–1212.

52. Prajapati, A.; Chhabra, J.K. Improving package structure of object-oriented software using multi-objective optimization and weighted class connections. J. King Saud Univ. Comput. Inf. Sci. 2017, 29, 349–364.

53. Deb, K.; Pratap, A.; Agarwal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comp. 2002, 6, 182–197. [CrossRef]

54. Kumari, A.C.; Srinivas, K. Hyper-heuristic approach for multi-objective software module clustering. J. Syst. Softw. 2016, 117, 384–401. [CrossRef]

55. Hofmeister, C.; Nord, R.; Soni, D. Applied Software Architecture; Object Technology Series; Addison Wesley: Boston, MA, USA, 1999.
56. Shtern, M.; Tzerpos, V. Clustering methodologies for software engineering. *Adv. Softw. Eng.* **2012**, *2012*. [CrossRef]

57. Bavota, G.; Gethers, M.; Oliveto, R.; Poshyvanyk, D.; Lucia, A.D. Improving software modularization via automated analysis of latent topics and dependencies. *ACM Trans. Softw. Eng. Method.* **2014**, *4*, 1–33. [CrossRef]

58. Mkaouer, W.; Kessentini, M.; Shaout, A.; Koligheu, P.; Bechikh, S.; Deb, K.; Ouni, A. Many-Objective Software Remodularization Using NSGA-III. *ACM Trans. Softw. Eng. Methodol.* **2015**, *24*, 1–45. [CrossRef]

59. Prajapati, A.; Chhabra, J.K. MaDHS: Many-objective Discrete Harmony Search to Improve Existing Package Design. *Comput. Intell.* **2019**, *35*, 98–123. [CrossRef]

60. Prajapati, A.; Chhabra, J.K. Improving modular structure of software system using structural and lexical dependency. *Inf. Softw. Technol.* **2017**, *82*, 96–120.

61. Prajapati, A.; Chhabra, J.K. Harmony search based remodularization for object-oriented software systems. *Comput. Lang. Syst. Struct.* **2017**, *47*, 153–169.

62. Prajapati, A.; Chhabra, J.K. FP-ABC: Fuzzy Pareto-Dominance Driven Artificial Bee Colony Algorithm for Many Objective Software Clustering. *Comput. Lang. Syst. Struct.* **2018**, *51*, 1–21.

**Publisher’s Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).