British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

Addressing Multi-objectives Search Challenges in Code Search Systems using CROPS Algorithms

Bassey Asuquo Ekanem
Computer Science Department, Delta State University of Science and Technology, Ozoro, Delta State, Nigeria

Kehinde K. Agbele
Computer Science Department, Elizade University, Illara-Mokin, Ondo State, Nigeria

doi: https://doi.org/10.37745/bjmas.2022.0444 Published March 19, 2024

Citation: Ekanem B.A. and Agbele K.K. (2024) Addressing Multi-Objectives Search Challenges in Code Search Systems using CROPS Algorithms, *British Journal of Multidisciplinary and Advanced Studies*: Engineering and Technology, 5(2),1-21

ABSTRACT: *One of the biggest challenge in software reuse is the huge amount of time and efforts required by re-users to evaluate the suitability of reusable components in a search before they are selected for reuse. This becomes more challenging especially where large number of Pareto solutions are generated based on multiple objectives of a reuse scenario. This could lead to wrong choice of components and low quality products where the re-user is not patient enough to evaluate the long list of partially ordered components presented by the code search engine. In addressing this challenge, many multi-objective evolutionary algorithm (MOEA) frameworks have been introduced namely non-dominated sorting genetic algorithms, MOEA based on decomposition, preference-based MOEAs and many others. In this research, a type of preference-based MOEA named CROPS (Components Ranking Optimization and Selection) Algorithm is presented. CROPS uses functional requirements and the preferential order of non-functional requirements together with high-level objectives for filtering and sub-ranking of components to generate distinctive ranks of Pareto sets based on components suitability. Using this approach, time and efforts required by a re-user to search, rank and select quality components for reuse in a given re-use scenario is minimized.*

KEYWORDS: preference-based MOEAs, components ranking optimization and selection (CROPS) algorithm, software reuse, Pareto solutions

INTRODUCTION

In most reuse scenarios, reusers spent huge amount of time and efforts to evaluate the suitability of reusable components from multiple repositories before they are selected for reuse. Also, where the number of Pareto solutions from multiple objectives of a reuse scenario is very large and presented without distinctive ranks, the problem could become too complex to handle thereby resulting in wrong choice of components if the re-user is not patient enough to evaluate them thoroughly. In recent times, code search engines have significantly address code search problems with respect to functional suitability of components for different reuse scenarios. However, the problem of components ranking

according to their non-functional suitability still remains a challenge due to inadequate weighting metrics to quantify multiple non-functional requirements needed for such processes (Kessel and Atkinson, 2016; Rizk-Allah et al., 2020; Özçevik, 2021).

To address this, the use of preference-based techniques has been presented by some researchers where the re-user is required to specify the preferential order of the non-functional properties to be used in components ranking rather than specifying quantitative weighting information for such properties since it is difficult for re-users to derive such quantitative values (Biswas and Suganthan, 2018; Arsene et al., 2019). Unfortunately, despite these efforts, the issue of distinctive ranking of components based on non-functional requirements especially with very large Pareto sets still remain unresolved as evident in huge amount of components presented by code search engines from a simple component search of a given re-use scenario (Arsene et al., 2019; Jha and Mishra, 2016; Cai et al., 2019).

In view of the above, this research proposes a technique for distinctive ranking of Pareto sets of components using Preference-based Multi-Objective Evolutionary Algorithm framework**.** The proposed technique is named, **Components Ranking Optimization and Selection Technique,** simply abbreviated as **CROPS Technique.** CROPS uses functional requirements and the preferential order of non-functional requirements (i.e. quality criteria) together with other high-level objectives for filtering and sub-ranking of components to generate distinctive ranks of Pareto sets based on components suitability. Using CROPS, best components can be easily selected from different repositories with minimum time and efforts.

RELATED WORKS

Multi-objective Evolutionary Algorithms have gained prominence in the last decade particularly in solving complex computing problems relating to components-based software engineering (CBSE). Wide adoption of these algorithms is due to the numerous benefits they provide particularly in finding optimal solution to multi-objective problems (Eita et al., Wang et al., 2017; Rajabi & Witt, 2020; Slowik & Kwasnicka, 2020). In ensuring continuous improvement of these algorithms, many research efforts have been undertaken. This section presents a brief review of these related works.

In Chung and Cooper (2004), an approach for ranking and selecting COTs for reuse based on functional and non-functional requirements is presented. This approached named CARE (COTS-Aware Requirement Engineering) was designed to improve reusers' confidence in selected COTS through a thorough evaluation of COTs using non-functional requirements (NFR) Framework. In Grau et al. (2004) a technique for COTs ranking and selection named DesCOTS (Description, Evaluation and Selection of COTS) is presented. The technique used ISO 9126 quality model and AHP technique in components ranking and selection. Cortellessa et al. (2008) demonstrated optimal choice of components in a multi-objective problem solving scenario using cost minimization as the objective function determined by reliability and delivery time constraints in a build-or-buy decisions. Kwong et al. (2010) presented an optimization model for COTS selection using genetic algorithm that is based on efficient measurement of cohesion and coupling of modules in COTs to be selected. Also, in Gupta et al. (2010), a technique for solving multi-objective optimization problem in components selection using fuzzy logic is introduced. The techniques which is aimed at increasing the product quality and reliability while reducing development cost and delivery time was formulated using fuzzy membership functions.

In Kumar et al (2012), the optimal choice of components in a fault-tolerance modular software system was demonstrated using maximization of system reliability and minimization of development costs as the objective function. Jha et al. (2014) introduced a fuzzy multi-objective approach for solving complex optimization problem involving multiple objective functions namely intra-coupling density (ICD), functionality, budget, and quality in choosing the optimal component from repositories. Also, Kessel and Atkinson (2016) presented an approach for ranking software components based on non-dominated sorting of components driven by relative importance of their non-functional properties as a partial ordering. The approach was supported with an algorithm and Code search engine designed and developed accordingly. In Ekanem and Woherem (2016), a method for extracting stable components from legacy systems for reuse is presented with non-functional characteristics used in components stability assessment.

In Zhang, Wang & Ye (2018), the new version of comprehensive particle swarm optimizer (CLPSO) called multi-objective complete learning particle swarm optimizer MOCLPSO is proposed While CLPSO could only solve non-dominated problems, MOCLPSO could solve both dominated and non-dominated problems. It is also capable of identifying better spread of solutions near the actual Pareto front and faster convergence to the true Pareto front than other algorithms. However, it's a time-consuming algorithm and takes more fitness evolution value to find the optimal solution of complex problems.
Yi et al. (2020) proposed a new heuristic-based multi-objective optimization algorithm called non-dominated sorting genetic algorithm II (NSGA-II). However, it can only solve non-dominated problems. Also, it uses more time and fitness evolution values to determine the optimal results in non-dominated problems. In Kalantari et al., (2021), a technique to optimize component selection problem through multi-objective optimization is presented. It maximizes the Fuzzy-Intra Coupling Density (Fuzzy-ICD) and functionality as objective function, while using account budget, delivery time, reliability, and Fuzzy-ICD as constraints of multi-objective problems. The proposed method was implemented using financial-accounting system as a case study which improved the selection process. Also, in Ekanem and Agbele (2021), a review of software components identification methods and quality assessment criteria was conducted which revealed the use of Quality Model for Object-oriented Design (QMOOD) as a major technique for components quality assessment amongst others.

Dou et al. (2021) proposed a new version of particle swarm optimizer (PSO) called multi-objective particle swarm optimizer (MOPSO) which enhanced the functions of PSO by integrating the concept of Pareto dominance which made it successful in solving dominated complex problems. However, MOPSO uses more fitness evolution values to determine the optimal results for a given multi-objective optimization problems. Elahi et al., (2022), proposed an extended version of a multi-objective group counselling optimizer called MOGCO-II. The proposed algorithm was demonstrated with Zitzler Deb Thieler (ZDT) function where it showed better performances compared with MOGCO, MOPSO, MOCLPSO, and NSGA-II. Also, it takes less fitness evolution value to find the optimal Pareto front.

FINDINGS FROM THE REVIEW

The review reveals abundance research efforts in components ranking and selection using MOEA frameworks. However, most of the reviewed approaches are based on components functional

requirements in a given reuse scenario. Furthermore, dealing with ranking and selecting problems that are based on non-functional properties of components are still challenging as much work is not done in this direction to address such concerns. More so, existing approaches using non-functional characteristics, at best only allow reusers to specify one non-functional objective in the query which in most cases, the reuser's interest may span multiple non-functional properties. Providing more methods, techniques and algorithms that could rank components based on multiple non-functional properties would therefore represent a significance step forward in enhancing software reuse.

RESEARCH METHODOLOGY

The research was conducted in the following stages:

i. Design the CROPS Algorithm
ii. Develop the model of CROPS Code Search Engine using the CROPS Algorithm.
iii. Populate the systems database with hypothetical software components.
A total of 60 hypothetical components were populated in the system. Twenty (20) components each for the following categories: Home Automation components (HA), Air pollution components (AP) and Weather Forecast components (WF). These components are designated with subscripts which indicates their unique identity. For instance, HA_1 and HA_7 represent the first and seventh Home Automation component respectively while WF_6 represents the sixth Weather forecast component.
iv. Use the Crops Code Search engine to search for component of interest based on specified reuse needs
v. Record the search results with respect to total number of components retrieved, number of components that meet the specified reuse needs and ranks of each component that meet the reuse needs in the retrieved list.

THE PROPOSED COMPONENTS RANKING OPTIMIZATION AND SELECTION METHOD

This research proposes a method named **Components Ranking Optimization and Selection (CROPS)** with a set of work packages needed to enhance the process of searching, ranking and selecting components based on re-user's preferred quality criteria. The method is so named because it's based on multi-objective optimization (MOP) technique. It is designed to utilize multiple functional requirements, multiple non-functional requirements and multiple preferred high-level objective criteria in the ranking and selection process. The preferred high-level objectives criteria are sub-divided into two categories namely high-level objectives for components filtering and high-level objectives for components sub-ranking.

The preferred quality characteristics are drawn from the 6 quality attributes of QMOOD Model (Bansiya and Davis, 2002) namely reusability, functionality, effectiveness, understandability, extendibility and flexibility) and 8 product quality attributes defined in ISO/IEC 25010:2011 quality model (ISO, 2017) namely functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability. The preferred high-level objectives for filtering include programming languages/software development tools, source of components and version while the high-level objectives for sub-ranking include number of downloads, cost of integration and release date.

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

A) Description of Stages in CROPS Method

Figure 1 presents the stages in the method which are explained thus:

i. Specification of Re-user's needs based on Re-use Scenario

In this stage, the re-user specifies requirements of the needed components for reuse scenario. Requirements to be specified include components functional requirements and non-functional requirements (i.e. quality characteristics). Both functional requirements and non-functional requirements are multi-valued entities, hence suitably implemented with the MOP technique. These needs are to be provided to the code search engine through the user interface (UI) designed to capture these inputs.

ii. Searching the Repositories for Matching Components

When the inputs are provided by the reuser, the code search engine searches the repositories/libraries for components that meet the specified criteria. The search method adopted in this case is the Clustering method. Where matching components are not found, the re-user has to review the specified inputs especially the functional requirements and repeat the search process. In event where matching components are found, the process proceeds with stage 3.

iii. Generation of Pareto Sets from Identified Components

Following a successful identifications of components that match the specified functional and non-functional requirements, the system applies the non-dominated sorting method in sorting the identified components which results in $P = [P_1 \ldots P_n]$ non-dominated sets where 1 to n represent a strict ordering of non-distinguishable candidate sets in ascending order. The ranking method adopted for this process is Pareto ranking of Non-dominated sorting technique. At this stage, the partial ranking of the components is obtained. Table 1 illustrates this approach with 21 components.

Table 1: Ranking of identified Non-dominated Components

Ranking	Non-dominated Sets (P_i)	Candidates
1	P_1	{c_2, c_6, c_7, c_8, c_9, c_{10}, c_{16}, c_{19}, c_{21}}
2	P_2	{c_1, c_3, c_5, c_{11}, c_{14}}
3	P_3	{c_4, c_{12}, c_{17}, c_{20} }
4	P_4	{c_{13}, c_{15}, c_{18}}

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

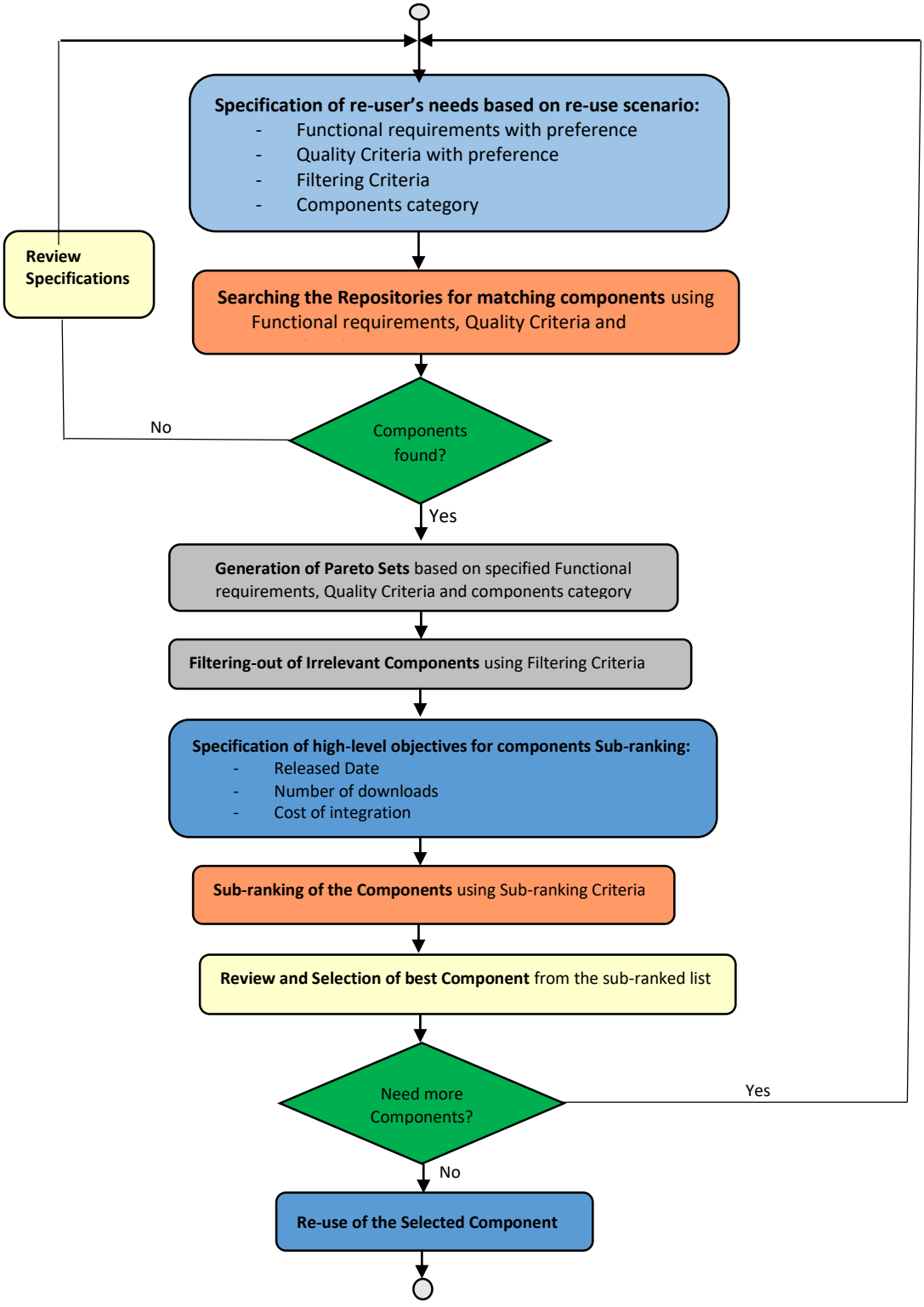

Fig. 1: UML Diagram Showing Components Ranking and Selection Process

In the above Table, P_1 is the Pareto front which contains the best components based on the re-use scenario, although their distinctive order is not specified by the Pareto ranking method. As mentioned earlier, where the number of P_i candidate sets is very large, much efforts and time will be required by the re-user to examine all the components manually to identify the best, hence the need for filtering and sub-ranking processes as indicated in steps four and five.

iv. Filtering-out of Irrelevant Components using Filtering Criteria

This stage is introduced to address situations where some components in the Pareto Front may be functionally and quality sufficient based on the specified criteria yet not relevant to the re-use scenario. As part of the optimization process, the code search engine should be able to identify and filter out such components. For instance, assuming the re-user needed components developed with Python language, but in the Pareto front, components developed with C and Java are also included because they are found to be functionally and quality sufficient, such components cannot be selected by the re-user because they are irrelevant to the re-use scenario and won't be useful at all. That's where filtering comes in. The filtering process will be done using high-level objectives for components filtering example programming language which are provided for in the User Interface (UI) for the user to interactively select from the list. By selecting Python for instance, all components developed with other languages will be automatically removed from the Pareto front.

vi. Specification of Preferred High-level Objectives for Components Sub-ranking

Sub-ranking of components is necessary to increase their distinctiveness since the components in the Pareto sets are usually partially ordered. To achieve this, preferred high-level objectives for components sub-ranking are needed which include number of downloads, costs of integration and release date. These parameters being part of the components definition are accessed by the code search engine and utilized accordingly. Priority is given to components with high number of downloads, low costs of integration and recent release dates.

vii. Sub-ranking of the Components

Using the specified preferred high-level objectives for components ranking the code search engine generates a list of optimized components with distinctive ranks in ascending order with the best occupying the topmost rank making it possible for the re-user to make his choice.

viii. Review and Selection of best Components for Reuse

Having presented the non-dominated components in their distinctive ranks revealing components that best fit the re-use scenario in terms of functional requirements, quality criteria and preferred high-level criteria for components filtering and sub-ranking, the re-user can now review the list and select the component considered best in the re-use scenario.

ix. Re-use of the Selected Components

The last stage in the process is to integrate the selected components into the re-use project to develop software system.

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

B) Parameters for Components Ranking and Selection

CROPS method uses the following parameters for components ranking optimization and selection: **functional requirements parameters, non-functional requirements parameters**, **filtering Parameters** and **sub-ranking Parameters.** A brief description of these parameters is given below.

i) Functional Requirement Parameters

Functional requirements parameters are parameters that represent a set of functions expected to be performed by the required component, based on the re-use scenario. For instance, the functional requirements of language translation component for a re-use project might be specified as follows: the components should be able to accept inputs in the form of text, voice and video; translate text inputs from one language to another; translate voice inputs from one language to another; translate video inputs from one language to another and measure the size of the input and output. These can be expresses as: a set of n specified functional requirements $SFR_1(C)$ … $SFR_n(C)$ defined for each component required by the re-user; where $SFR_1(C)$ refers to the first functional requirement while $SFR_n(C)$ is the last functional requirement in the set. Similarly, the functional parameter can be designated as, SFR(n).

From the functionality point of view, assessment of a component based on the specified functional parameters could result in three possibilities, namely *functional sufficient* – if a component completely fulfils the functional requirements of the re-user; *functional insufficient* – if a component does not meet all the functional requirements of the re-user; and *functional superfluous* - if a component provides more functionality than is actually needed in a particular reuse scenario (Kessel and Atkinson 2015). Therefore, the ideal component from the point of functionality assessment is a components that is functional sufficient and without superfluous functionality**.**

ii) Non-functional Requirement Parameters

Non-functional requirement parameters representing the set of constraints imposed on the functional requirements which are to be met by the component as expressed by the re-user. Simply put, these parameters are used to represent the quality characteristics of components needed for re-use. These include *functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability* which are specified in ISO/IEC 25010:2011 (ISO, 2017). For instance, the re-user may expect that, the language translation component to be select for re-use should possess the following five quality characteristics: maintainability, functional suitability, Usability, Security and Portability.

Accordingly, these can be expresses as:
a set of m specified Quality Attributes $SQA_1(C)$ … $SQA_m(C)$ defined for each component required by the re-user; where $SQA_1(C)$ refers to the first quality attribute and $SQA_m(C)$ refers to the last quality attribute in the set. From the above specification $SQA_1(C)$ = Maintainability, $SQA_2(C)$ = functional suitability, $SQA_3(C)$ = Usability, $SQA_4(C)$ = Security and $SQA_5(C)$ = Portability. Therefore, the Quality requirement parameter can be designated as, SQA(m).

It is worth mentioning that, the order of these quality criteria is important as it directly affects the ranking of the components, hence the re-user is expected to outline them in their order of priority in the space provided in the systems UI. Just as the case with functional parameters, assessment of components in

terms of quality criteria could result in components that are *quality sufficient* – component that completely fulfils the quality requirements of the re-user; *quality insufficient* – component that does not meet all the quality requirements of the re-user; and *quality superfluous* - component that provides more functionality than are actually needed by the re-user in a particular reuse scenario. Therefore, the ideal component from the quality requirements point of view is a components that is quality sufficient and has no superfluous quality attributes.

iii) High-level Objective Parameters

This set of parameters are those used to achieve distinctiveness among the Pareto sets. They are divided into two groups, namely filtering parameters and sub-ranking parameters. As the name implies, filtering parameters are used to filter out components that may be part of the Pareto sets but not relevant to the reuse scenario while sub-ranking parameters are used in sub-ranking the components in the Pareto front. For instance, as explained earlier, if the user is interested in components developed with a particular programming language say Python, then there is no need including components developed with other programming languages in the Pareto set. Hence, this parameter is used to represents high level objectives like programming language, source of component and version of the components to enable the code search engine filter out irrelevant components from the Pareto set.

From the above, components filtering parameters can be expresses as a set of p specified High-level Objectives $SHF_1(C)$ … $SHF_p(C)$ defined for each component; where $SHF_1(C)$ refers to the first specified high-level objective and $SHF_p(C)$ refers to the last specified high-level objective in the set. Therefore, the filtering parameter can be designated as, SHF(p). Using the above specified order, $SHF_1(C)$ = Programming language, $SHF_2(C)$ = source of component, and $SHF_3(C)$ = version of the component.

For the sub-ranking parameters, they are used to ensure that there are some level of distinctiveness amongst the retrieved components. For instance, components in a Pareto front that share the same partial rank, can be made distinctive using objectives like number of downloads, released date and cost of integration since these objectives will rarely be similar for any two components in the Pareto set. However, not ruling out the possibility of same occurrences, where it happens, the re-user can examine such components with similar occurrences and make a choice easily since such occurrences may be very few and easier to review. In view of the above, components sub-ranking parameters can be expressed as set of q specified High-level Objectives, $SHS_1(C)$ … $SHS_q(C)$ defined for each component required by the re-user; where $SHS_1(C)$ refers to the first specified high-level objective and $SHS_q(C)$ refers to the last specified high-level objective in the set.

Using the above specified order, $SHS_1(C)$ = number of downloads, $SHS_2(C)$ = release date, and $SHS_3(C)$ = cost of integration. It is worth mentioning that, the order of the high-level objectives criteria (i.e. filtering and sub-ranking) is important as it directly affects the filtering and sub-ranking of the components, hence the re-user is expected to present these requirements and their order of priority via the interactive interface of the code search engine.

FORMULATION OF MATHEMATICAL MODEL AND DEFINITIONS:

The components ranking problem to be address by the algorithm in this research in viewed as an optimization problem. However, since there are usually many functional and many non-functional

criteria to be satisfied, the optimization problem is characterized as a multi-criteria optimization problem. For effective ranking of components, a type of *non-dominated sorting Techniques* called *Multi-objective Evolutionary Technique* that uses *Preference-based Multi-objective Evolutionary Algorithm* framework to solve multi-objective problem (MOP) is adopted. Using this technique, a non-dominated set like the set of best global solutions on the Pareto frontier that contains all solutions that are not dominated by other solutions with respect to a given set of objectives will be derived from which high-level objectives will be applied to obtain the sub-ranks of the components to discriminate components within a non-dominated sets thereby making it easier to select the optimal component. In non-dominated sorting, *a candidate A is said to dominate another candidate B,* if and only if

i. there is no objective of A worse than that objective of B and
ii. there is at least one objective of A better than that objective of B.

From the above, mathematically, the multi-objective problem is represented as:

Minimize: $F(x) = [f_1(x), f_2(x), \ldots, f_M(x)]$ ---------- (1)

Subject to: $x \in \Omega$

Where, Ω is the decision vector (variable) space, M is the number of objectives. A component $x^* \in \Omega$ is a Pareto Optimal if there is no component $x \in \Omega$ such that $F(x)$ dominates $F(x^*)$. $F(x^*)$ is called Pareto Optimal (objective) vector. In other words, any improvement in the Pareto Optimal point in one objective must lead to deterioration of at least one other objective. The set of all the Pareto Optimal Components is called Pareto set (PS) and the set of all the Pareto optimal objective vectors is the Pareto front (PF).

a) Non-dominance

For a set of N specified functional requirements $SFR_1(C) \ldots SFR_n(C)$, set of M specified Quality attributes $SQA_1(C) \ldots SQA_m(C)$ and set of q specified Filtering criteria $SHF_1(C) \ldots SHFq(C)$ defined for each component C of a set Ω of components, and all of them being maximized, we say that C_1 dominates C_2 If :

i. $SFR_n(C_1) \geq SFR_n(C_2)$ for all 1, …N and $SFR(C_1) = (SFR_1(C_1), \ldots, (SFR_n(C_1) \neq SFR(C_2)$
ii. and $SQA_m(C_1) \geq SQA_m(C_2)$ for all 1, …M and $SQA(C_1) = (SFR_1(C_1), \ldots, (SQA_m(C_1) \neq SQA(C_2)$.
iii. and $SHF_q(C_1) \geq SHF_q(C_2)$ for all 1, …Q and $SHF(C_1) = (SHF_1(C_1), \ldots, (SHF_q(C_1) \neq SHF(C_2)$.

Simply put, for a given set Ω, a component C* is said to be non-dominated if there is no other component dominating C*.

b) Sub-ranking

For a set of n specified High-level objectives $SHR_1(C) \ldots SHR_n(C)$ defined for each non-dominated component C* of non-dominated set P_i, and all of them being maximized, we say that C^*_1 is of a higher rank than C^*_2 If $SHR_n(C^*_1) > SHR_n(C^*_2)$ for all 1, …N and $SHR(C^*_1) = (SHR_1(C^*_1), \ldots, (SHR_n(C^*_1) \neq SHR(C^*_2)$

THE PROPOSED COMPONENTS RANKING AND SELECTION ALGORITHM:

Having described the CROPS method and the supporting mathematical model, this section presents a brief description of the CROPS Algorithm – a form of Preference-based Multi-objective Evolutionary

Algorithms. The aim of the algorithm is to ensure that components included in the Pareto sets are those that are most relevant to the re-use scenario and are ranked in an order that best meet the re-user's needs. The components ranking problem to be address by the algorithm viewed as an optimization problem involving many functional and non-functional criteria to be satisfied, hence characterized as a multi-criteria optimization problem. For effective ranking of components, the proposed algorithm is based on non-dominated sorting algorithm instead of weight-based algorithms since users are not required to provide specific objective weightings. With this, a non-dominated set like the set of best global solutions on the Pareto frontier that contains all solutions that are not dominated by other solutions with respect to a given set of objectives will be derived from which the best can easily be selected. Further description of the algorithm follow thus:

A) Assumptions for the Algorithm

The following assumptions are made concerning the proposed algorithm:

i. unless specified otherwise, functional sufficiency is the most important criteria in the ranking algorithm because it's the primary requirement for components identification without which no component will be available for ranking.
ii. The code search engines are capable of performing automated analysis of code to measure quality attributes like reusability, modularity and others which are used in preference-based ranking of the components.
iii. The reusable components are properly catalogued following specified standard to guarantee easy assessment and generation of Pareto sets based on specified re-use scenario.
iv. Information on high-level objective criteria are provided by the re-user using posteriori and interactive methods through an interactive interface of the code search engine with their preferences adequately accounted for.

B) Design of the Algorithm

The proposed CROPS Algorithm for ranking and selection software components is given thus:

Algorithm: Components Ranking Optimization and Selection (CROPS) Algorithm

---

1: // CROPS Algorithm
2: // Algorithm to rank and select optimized components from repositories given
3: // a set of specified **functional requirements**, **Quality Attributes** and **high-level objectives**
4: Input: Set of specified functional requirements $SFR_n(C)$ specified on component C i.e. ($SFR_1(C)$ … $SFR_n(C)$
5: Input: Set of specified Quality Attributes $SQA_m(C)$ specified on component C i.e. ($SQA_1(C)$ … $SQA_m(C)$
6: Input: Set of specified High-level Objectives for Filtering $SHF_n(C)$ on Comp.
7: i.e. ($SHF_1(C)$ … $SHF_n(C)$ ≡ {programming language, source of component, version}
8: Input: Set of specified High-level Objectives for Sub-ranking $SHS_n(C)$ on Comp.
9: i.e. ($SHS_1(C)$ … $SHS_n(C)$ ≡ {cost of integration, number of downloads}
10: Input: Set of Components $C_n(R)$ in repositories i.e. ($C_1(R)$ … $C_n(R)$
11: Output: Set of non-dominated Components **P**(R) selected from repositories i.e. **P**(R) … **P**(R)
12: Output: Optimized Component C_{opt} (R) selected from the set of non-dominated components
13: Require: C_n of length > 0, SFR_n of length > 0, SQA_m of length > 0, SHO_n of length > 0)
14: Begin

```
15:    compCategory ← getcategory (Comp)  // input components category
16:    SFRn ← getPrimaryCriteria (functionalReq(i), lastFuncReq)
17:    SQAm ← getSecondaryCriteria (QualityReq(i), lastQaulityReq)
18:    SHFn ← getFilteringCriteria (filteringReg(i), lastfilteringReq)
19:     function rankAndSELECT(Cn, compCategory, SFRn, SQAm, SHFn, SHSn lastPriority)
20:        P← nonDominatedSORT (Cn, compCategory, SFRn, SQAm, SHFn)
21:         qualityPriority ← NEXTQualityPriority (SQAm, lastPriority)
22:         If qualityPriority == -1 then          // No more quality priorities left
23:         SHSn ← getSubrankingCriteria (SHSn, subRankPriority)
24:         subRankingPriority ← NEXTSubRankingPriority (SHSn, lastRankingPriority)
25:         If subRankingPriority == -1 then      //  No more quality priorities left
26:                Return join (P)                //   to one-dimensional array
27:          For pi in P Do
28:          If length (pi) > 1 then                    //  Sub-ranking only for sets >1
29:          New Pi ← rankAndSELECT (pi, compCategory, SFRn, SQAm, SHOn,
30:                subRankPriority)              //  Recursive Call of rankAndSELECT
31:          pi ← join(new Pi)                  // to one-dim. array of distinctive ranked components
32:              Return join(P)                 // to one-dim. array of distinctive ranked components
33:              Copt ← getSelectedComponent (P)   // the selected component by the re-user
34:          Require: Cn of length > 0, SFRn of length > 0, SQAm of length > 0
35:          function nonDominatedSort(Cn, compCategory, SFRn, SQAm, SHFn)
36:              P ← [ ]                            //   Non-dominated sets in ascending order
37:              p ← Cn                             //   Current dominated set
38:              while length (p) > 0 Do
39:               n ← nonDominatedSet(p, compCategory, SFRn, SQAm, SHFn,) // Partial Ordering
40:               P ← APPEND(P, n)                  //  add n to P
41:               p ← p\n                           //  set p to dominated candidates
42:               Return P
43:  End.
```

In designing the algorithm, the basic version of non-dominated sorting algorithm is used with some enhancements inline with CROPS properties as described earlier. CROPS algorithm uses two functions namely **nonDominatedSort** and **rankAndSELECT** functions, where **rankAndSELECT** is invoked recursively.to rank and sub-rank components using quality criteria and high-level objectives for sub-ranking respectively. Inputs into the Algorithm namely components category, functional requirements, quality criteria and high-level objectives for filtering are entered into the algorithm through the search engine interface as indicated in lines 15 to 18.

Thereafter, the function, **nonDominatedSort** is called in line 20 to execute and generate the non-dominated sets from a set of components found in various repositories using components category, functional requirements, quality criteria and high-level objectives for filtering as specified by the re-

user (see lines 35 to 41). However, the candidates in each non-dominated sets at this stage are non-distinguishable since they are presented in a partial order using the specified criteria.

Distinctive ranking of the components is achieved using quality priority and sub-ranking priority (see lines 27 to 31). Using the quality priority, the components are ranked based on the quality interest of the re-user for the re-use scenario while the sub-ranking priority are used for distinctive ranking of components. For instance, components in the Pareto Sets may be of equal standing in terms of the quality indicators even to the extent of the quality priority specified by the reuse say accuracy, availability, portability, security and maintainability (in that order). To break the tie, number of downloads and cost of integration of each component will be used to determine which should occupy highest position in the rank and so forth since the probability of two components having the same number of downloads and cost of integration is low. Following the sub-ranking of components, the re-user can then make his choice from the topmost ranked component(s) as indicated in line 33.

SYSTEM DEVELOPMENT AND VALIDATION:

A model of the CROPS Code search engine was developed and validated using data generated for the research purpose. A brief description of the system and the validation process follow thus:

a) Description of the CROPS Code Search Engine

The CROPS search engine has a Main user interface through which the user interact with the system. Figure 2 shows the main user interface of the CROPS code search engine.

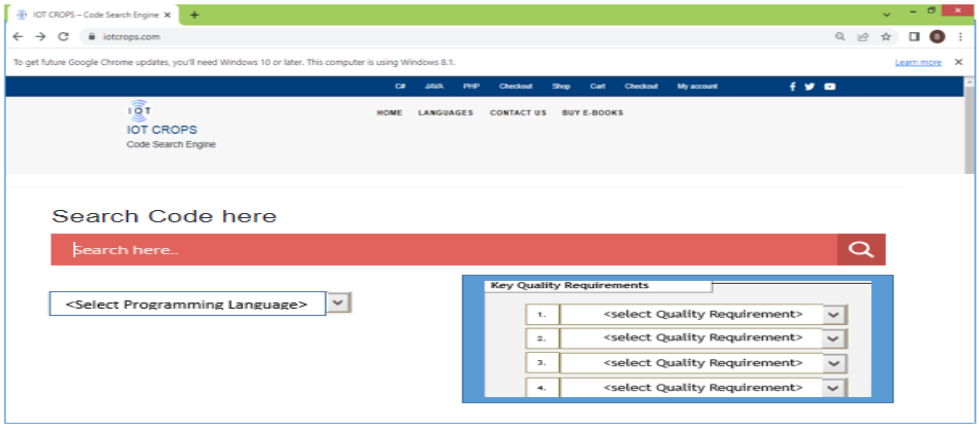

Figure 2: The Main User Interface of the CROPS Code Search Engine

On the interface, the space labeled **Search here…** allows the user to enter text that briefly describes the components he intends to search. The search text contains basic elements for functional requirements and components category which are easily extracted by the Code Search engine. The section indicated as key quality requirements allows the user to specify the non-functional characteristics of the components in the preferred order. For instance, a user may be interest in a component with reusability as the highest priority followed by modularity, extensibility and so forth. In this case, he can set this order of priority by selecting reusability in the first box, followed by modularity in the second box then extensibility in the third box and so forth. It is worth mentioning that, its this order that will determine the ranking of the components in the multi-objective optimization system, hence must be accurately specified by the reuser.

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

b) Dataset for Validation

Table 2 shows the dataset used for the systems validation.

Table 2: Dataset for Validation

Component	QMOOD indices (%)						technology	language
	modularity	reusability	extensibility	Understand-bility	functionality	effectiveness		
HA_1	88	77	91	85	81	90	Raspberry pi	Python
HA_2	76	79	90	65	84	66	Arduino	Jupyter Notebook
HA_3	91	88	79	78	66	81	Raspberry pi	Python
HA_4	92	90	81	88	84	89	Raspberry pi	Python
HA_5	69	86	88	90	81	76	Arduino	C++
HA_6	84	73	65	75	90	86	Arduino	Java
HA_7	73	78	89	80	96	68	Arduino	Jupyter Notebook
HA_8	69	81	83	88	78	79	Raspberry pi	Python
HA_9	81	70	66	69	88	88	Arduino	Java
HA_{10}	93	69	89	71	90	90	Raspberry pi	C++
HA_{11}	79	81	76	91	84	79	Arduino	Jupyter Notebook
HA_{12}	89	93	87	95	74	96	Raspberry pi	Python
HA_{13}	90	69	80	69	78	88	Arduino	Kotlin
HA_{14}	88	81	85	79	76	78	Raspberry pi	Python
HA_{15}	65	73	85	90	66	79	Arduino	Python
HA_{16}	90	63	87	93	69	88	Raspberry pi	C++
HA_{17}	88	80	69	79	82	81	Arduino	Python
HA_{18}	71	69	88	88	90	63	Raspberry pi	Python
HA_{19}	89	86	89	90	81	88	Arduino	C++
HA_{20}	88	65	92	83	90	81	Arduino	Python
AP_1	69	80	71	73	67	76	Arduino	Python
AP_2	65	88	78	93	88	90	Raspberry pi	Dart
AP_3	84	69	90	79	78	81	Raspberry pi	Jupyter Notebook
AP_4	74	71	76	88	83	75	Raspberry pi	Kotlin
AP_5	69	91	79	77	90	77	Arduino	JavaScript
AP_6	88	85	88	86	66	80	Raspberry pi	C++
AP_7	79	81	69	73	78	69	Arduino	Jupyter Notebook
AP_8	80	90	75	63	90	90	Raspberry pi	Kotlin
AP_9	89	96	88	79	66	81	Arduino	Kotlin
AP_{10}	88	78	81	81	84	82	Raspberry pi	JavaScript
AP_{11}	78	88	79	90	81	76	Raspberry pi	Jupyter Notebook
AP_{12}	84	72	85	76	67	88	Arduino	Python
AP_{13}	90	84	79	91	75	81	Arduino	Jupyter Notebook
AP_{14}	69	88	88	69	71	90	Raspberry pi	Java
AP_{15}	87	78	72	82	67	83	Raspberry pi	Python
AP_{16}	90	90	69	90	81	65	Raspberry pi	Python
AP_{17}	88	68	82	82	75	84	Raspberry pi	Kotlin
AP_{18}	69	79	90	76	81	66	Arduino	Jupyter Notebook
AP_{19}	80	88	88	73	77	69	Arduino	C++
AP_{20}	78	79	69	66	68	75	Raspberry pi	Kotlin
WF_1	81	81	67	93	78	79	Raspberry pi	Python
WF_2	85	90	81	79	88	80	Raspberry pi	JavaScript
WF_3	78	67	78	88	90	89	Raspberry pi	C++
WF_4	66	88	77	90	89	81	Raspberry pi	Java
WF_5	87	78	78	86	67	90	Arduino	Jupyter Notebook
WF_6	91	81	65	73	79	84	Raspberry pi	Kotlin
WF_7	89	90	90	93	88	91	Raspberry pi	Jupyter Notebook
WF_8	83	66	86	79	69	88	Raspberry pi	C++
WF_9	67	78	89	88	65	67	Raspberry pi	Python
WF_{10}	69	90	78	55	84	89	Arduino	Jupyter Notebook
WF_{11}	88	81	88	86	74	76	Raspberry pi	Kotlin
WF_{12}	91	84	76	73	87	65	Arduino	Jupyter Notebook
WF_{13}	80	81	69	87	88	78	Raspberry pi	C++
WF_{14}	78	67	89	79	79	77	Arduino	Java

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

WF_{15}	**85**	**80**	**88**	**91**	**80**	**89**	Raspberry pi	Python
WF_{16}	**87**	**69**	**79**	**72**	**89**	**69**	Raspberry pi	Kotlin
WF_{17}	**81**	**80**	**90**	**78**	**85**	**82**	Arduino	Jupyter Notebook
WF_{18}	**79**	**92**	**88**	**73**	**78**	**91**	Raspberry pi	C++
WF_{19}	**89**	**71**	**56**	**88**	**90**	**79**	Arduino	Java
WF_{20}	**92**	**79**	**77**	**71**	**85**	**87**	Arduino	Python

The data used in this research were randomly generated and used to represent 20 reusable components each in three categories namely Home Automation components (HA), Air Pollution components (AP) and Weather Forecast Component (WF) as shown in Table 1. Also, the non-functional attributes of these components are expressed in terms of QMOOD model (Bansiya and Davis, 2002; Özçevik, 2021) with the six QMOOD indices namely Reusability, Extendibility, Flexibility, Functionality, Understandability and Effectiveness indicated. From the table, the modularity of H_1 for instance is expressed as 88% while functionality is 81%.

c) Setting the Search Parameters

To conduct a search, the required inputs must be provided to the code search engine. As indicated in Figure 3, the reuser is interested in components for home automation system implemented on Raspberry Pi. This expression serves as functional requirements parameter, where the multiple requirements namely *home automation system, Raspberry Pi* and *detect intruders* are extracted by the code search engine and used accordingly.

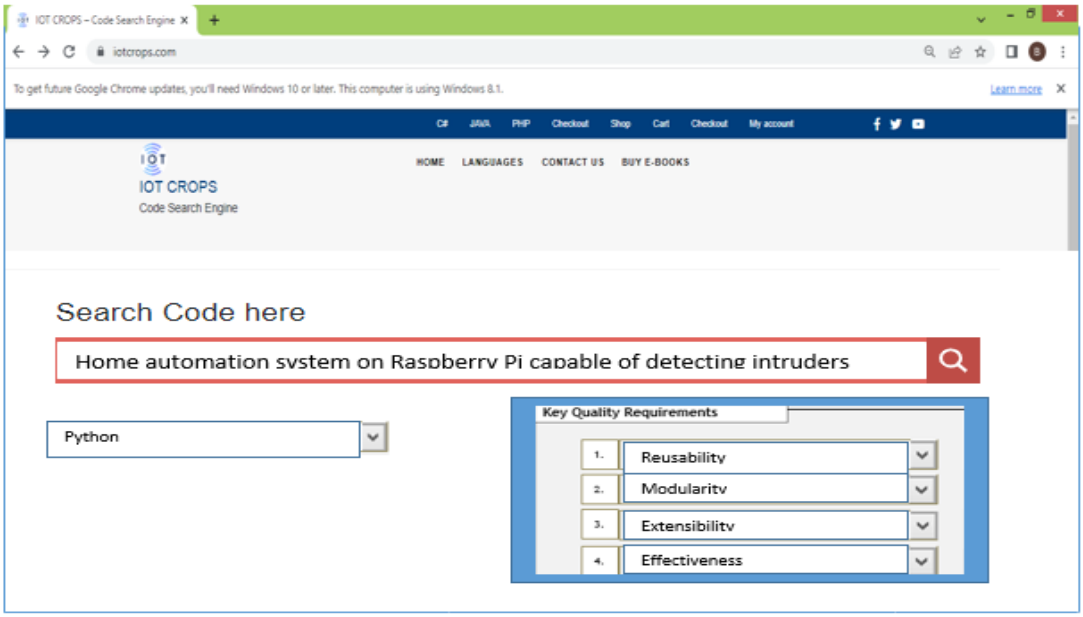

Figure 3: Setting up a Search in CROPS Code Search Engine

In terms of the non-functional characteristics, the reuser is interested in the multiple criteria namely reusability, modularity, extensibility and effectiveness given in the specified order of priority. Finally, the filtering parameter is set to Python language to enable the system filter out components not satisfying this criteria. The result of this search process is presented in table 4.

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

RESULTS AND DISCUSSIONS

Based on the functional requirements, non-functional characteristics, filtering and sub-ranking parameters provided by the reuser, the search result from the Code search engine is shown in Table 4.

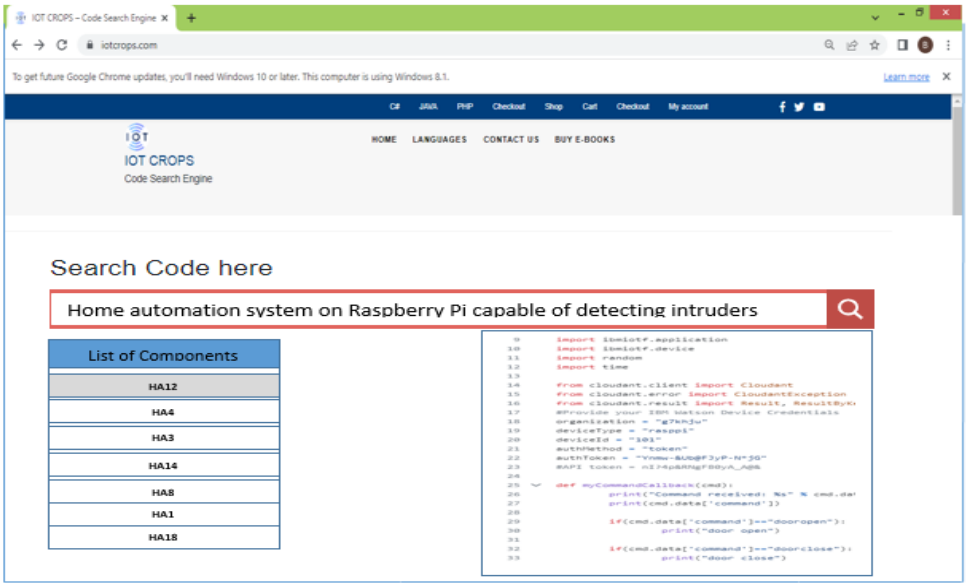

Figure 4: Search Results from CROPS Code Search Engine

From the list of 60 components in the repository, the search is narrowed down to 20 components in the category of components i.e. Home Automation Components as specified by the reuser. By applying the functional requirement "Raspberry Pi", the list is further reduced to 9 components eliminating other that are based on other technologies. Applying Python as a filter reduces the identified components to 7 as indicated in Table 3. Furthermore, the impact of the sub-ranking parameter is observed in HA_{14} and HA_8 that have similar reusability value of 81% and their modularity values are used in sub-ranking them. In this case, HA_{14} with higher modularity value of 88% is placed above HA_8 which value is 69%.

Table 3: Search results based of the given Multi-objective criteria with Reusability as key quality criteria

Component	modularity	reusability	extensibility	Understandability	functionality	effectiveness	technology	language
HA_{12}	89	93	87	95	74	96	Raspberry pi	Python
HA_4	92	90	81	88	84	89	Raspberry pi	Python
HA_3	91	88	79	78	66	81	Raspberry pi	Python
HA_{14}	88	81	85	79	76	78	Raspberry pi	Python
HA_8	69	81	83	88	78	79	Raspberry pi	Python

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

HA_1	**88**	**77**	**91**	**85**	**81**	**90**	Raspberry pi	Python
HA_{18}	**71**	**69**	**88**	**88**	**90**	**63**	Raspberry pi	Python

The above illustration shows that this approach is capable of narrowing down the search results to few optimal solutions from the Pareto sets that could be selected with minimal time and effort.

The search was repeated with different variations in preferred order of the quality attributes which showed interesting results. By varying the order of the non-functional parameters, results presented in Tables 4 -6 were obtained, which indicate the strength of each component with respect to variable quality scenario which could further boast the confidence of the reuser in whatever component is eventually selected. The summary of these variations is presented in Table 7

Table 4: Search results based of the given Multi-objective criteria with Modularity as key quality criteria

Component	Modularity	reusability	extensibility	understandability	functionality	Effectiveness	technology	language
HA_4	**92**	**90**	**81**	**88**	**84**	**89**	Raspberry pi	Python
HA_3	**91**	**88**	**79**	**78**	**66**	**81**	Raspberry pi	Python
HA_{12}	**89**	**93**	**87**	**95**	**74**	**96**	Raspberry pi	Python
HA_{14}	**88**	**81**	**85**	**79**	**76**	**78**	Raspberry pi	Python
HA_1	**88**	**77**	**91**	**85**	**81**	**90**	Raspberry pi	Python
HA_{18}	**71**	**69**	**88**	**88**	**90**	**63**	Raspberry pi	Python
HA_8	**69**	**81**	**83**	**88**	**78**	**79**	Raspberry pi	Python

Table 5: Search results based of the given Multi-objective criteria with Effectiveness as key quality criteria

Component	modularity	reusability	extensibility	understandability	functionality	Effectiveness	technology	language
HA_{12}	**89**	**93**	**87**	**95**	**74**	**96**	Raspberry pi	Python
HA_1	**88**	**77**	**91**	**85**	**81**	**90**	Raspberry pi	Python
HA_4	**92**	**90**	**81**	**88**	**84**	**89**	Raspberry pi	Python
HA_3	**91**	**88**	**79**	**78**	**66**	**81**	Raspberry pi	Python
HA_8	**69**	**81**	**83**	**88**	**78**	**79**	Raspberry pi	Python
HA_{14}	**88**	**81**	**85**	**79**	**76**	**78**	Raspberry pi	Python
HA_{18}	**71**	**69**	**88**	**88**	**90**	**63**	Raspberry pi	Python

British Journal of Multidisciplinary and Advanced Studies:
Engineering and Technology, 5(2),1-21, 2024
Print ISSN: 2517-276X
Online ISSN: 2517-2778
https://bjmas.org/index.php/bjmas/index
Published by the European Centre for Research Training and Development UK

Table 6: Search results based of the given Multi-objective criteria with Extensibility as key quality criteria

Component	modularity	reusability	extensibility	understandability	functionality	Effectiveness	technology	language
HA_1	88	77	91	85	81	90	Raspberry pi	Python
HA_{18}	71	69	88	88	90	63	Raspberry pi	Python
HA_{12}	89	93	87	95	74	96	Raspberry pi	Python
HA_{14}	88	81	85	79	76	78	Raspberry pi	Python
HA_8	69	81	83	88	78	79	Raspberry pi	Python
HA_4	92	90	81	88	84	89	Raspberry pi	Python
HA_3	91	88	79	78	66	81	Raspberry pi	Python

Table 7: Summary of Components Ranking Occurrences

Component	No. of occurrences by Ranks		
	1st	2nd	3rd
HA_{12}	2 (reusability and effectiveness)	Nil	2
HA_3	Nil	1	1
HA_4	1 (modularity)	1	1
HA_8	Nil	Nil	nil
HA_{10}	Nil	Nil	nil
HA_1	1 (extensibility)	1	nil
HA_{14}	Nil	Nil	nil
HA_{16}	Nil	Nil	nil
HA_{18}	Nil	1	nil

Table 7 shows that, component HA_{12} dominated in two instances namely reusability and effectiveness, HA_4 and H_1 dominated in one instance each which are modularity and extensibility respectively. Using the above search results, HA_{12} is best for the reuse scenario. This clearly shows that using CROPS, time and efforts required by the reuser to search, rank and select components for reuse in any given reuse scenario is minimized while guaranteeing the choice of best components.

CONCLUSION

The quality of software reuse depends on the quality of reusable components selected from components repository for reuse. The task of getting quality and suitable components based on reuse scenario is

always a difficult one especially when the number of Pareto solutions in a MOP is very large and presented with conflicting objectives. The use of preference-based MOEAs has been of great benefits in addressing these challenge. Unfortunately, most of these methods do not address search concerns involving non-functional requirements (i.e. quality criteria), hence reusers' interests in this regard are usually not accommodated. CROPS (Components Ranking Optimization and Selection) algorithm - a type of preference-based MOEA i**s** designed to address this challenge. CROPS uses functional requirements and the preferred order of non-functional requirements (i.e. quality criteria) together with high-level objectives for filtering and sub-ranking of components to generate distinctive ranks of Pareto sets identified in a search. Using this approach, time and efforts required by the re-user to search, rank and select components for reuse in a given re-use scenario is minimized. Moreover, CROPS will enable reusers to select quality components from a wide range of components thereby enhancing software reuse.

RECOMMENDATIONS:

The following recommendations are necessary:

i. Implementation of CROPS Algorithm in code search engines is highly recommended as a means of addressing code search challenges involving non-functional characteristics of components.

ii. A standardized method for computing the values of quality attributes beyond those supported by QMOOD is highly recommended to cover a wide range of quality attributes that reusers may be interested in.

REFERNCES

Arsene, K. K. I., Adama, S., Kouadio, K. and Konam, B. (2019). Proposal of Automatic Methods for the Reuse of Software Components in a Library; International Journal of Advanced Computer Science and Applications
https://www.semanticscholar.org/paper/Proposal-of-Automatic-Methods-for-the-Reuse-of-in-a-Arsene-Adama/ec1add63da0af493d1bc4232c5686ef896cfcda0

Bansiya, J., Davis, C. G.: A hierarchical model for object-oriented design quality assessment. IEEE Transactions on software engineering, 28(1): pp. 4-17 (2002)

Biswas, P. and Suganthan, P. N. (2018). Multiobjective Evolutionary Optimization;
Wiley Encyclopedia of Electrical and Electronics EngineerinDOI:10.1002/047134608X.W8380

Cai et al. (2019) Cai X, Wang P, Du L, Cui Z, Zhang W, Chen J. Multi-objective three-dimensional DV-hop localization algorithm with NSGA-II. *IEEE Sensors Journal*. 2019;19(21):10003–10015. doi: 10.1109/JSEN.2019.2927733.

Chung, L. and Cooper, K. (2004). Matching, ranking and selecting Components: A COTS-aware requirements engineering and software architecting approach. In *Proceedings of the International Workshop on Models and Processes for the Evaluation of COTS Components at 26th International Conference on Software Engineering, (Edinburgh, Scotland, UK),* pp. 41-44

Cortellessa V, Marinelli F, Potena P (2008) An optimization framework for "build-or-buy" decisions in software architecture. Comput Oper Res 35(10):3090–3106

Dou et al. (2021) Dou J, Li J, Xia D, Zhao X. A multi-objective particle swarm optimisation for integrated configuration design and scheduling in reconfigurable manufacturing

system. *International Journal of Production Research.* 2021;59(13):3975–3995. doi: 10.1080/00207543.2020.1756507.

Elahi I, Ali H, Asif M, Iqbal K, Ghadi Y, Alabdulkreem E. (2022). An evolutionary algorithm for multi-objective optimization of freshwater consumption in textile dyeing industry. PeerJ Comput Sci. 2022 Mar 22;8:e932. doi: 10.7717/peerj-cs.932. PMID: 35494829; PMCID: PMC9044317.

Eita, Shoukry & Iba (2014) Eita M, Shoukry A, Iba H. Constrained group counseling optimization. Artificial Life Conference Proceedings 14; Cambridge: MIT Press; 2014.

Ekanem, B. A. and Woherem, E. (2016). Legacy Components Stability Assessment and Ranking using Software Maturity Index; International Journal of Computer Applications, 134(13), 22-30

Ekanem, B. A. and Agbele, K. K. (2021). A review of Software Component Identification Methods and Quality Assessment Criteria; European Journal of Applied Science 9(5), 194-209

Grau, G., Carvallo, J. P., Franch, X., and Quer, C. (2004). DesCOTS: a software system for selecting COTS components. In *Euromicro Conference, 2004 Proceedings; 30th* IEEE; pp. 118-126

Gupta P, Mehlawat MK, Verma S (2012) COTS selection using fuzzy interactive approach. Optim Lett 6(2):273–289

ISO (2017). **ISO (2017). ISO/IEC 25010:2011** - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models
https://www.iso.org/standard/35733.html

Jha, P. et al (2014). Optical component Selection based on Cohesion and Coupling for Component-based Software System under build-or buy Scheme; Journal of Computing Science 5(2): 233-242

Jha, S. K. and Mishra, R. K. (2016). Multi Criteria Based Retrieval Techniques For Reusable Software Components From Component Repository; International Journal of Engineering Applied Sciences and Technology, 2016 Vol. 1, Issue 6, ISSN No. 2455-2143, Pages 88-91 Published Online April - May 2016 in IJEAST (http://www.ijeast.com)

Kalantari, S., Motameni, H., Akbari, E. & Rabbani, M. (2021). Optimal components selection based on fuzzy-intra coupling density for component-based software systems under build-or-buy scheme. *Complex Intell. Syst.* **7**, 3111–3134 (2021). https://doi.org/10.1007/s40747-021-00449-z

Kessel, M. and Atkinson, C. (2015). Ranking Software Components for Pragmatic Reuse, 2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics, 2015

Kessel, M. and Atkinson, C. (2016). Ranking Software Components for reuse based on non-functional properties; Information system Frontier; Springer Science, New York 18:825-853
doi 10.1007/s10796-016-9685-3

Kumar D et al (2012). Optimal component selection problem for COTS based software system under consensus recovery block scheme: a goal programming approach. Int J Comput App 47(4):0975–1888

Kwong, C. K., Mu, L. F., Tang, J. F. and Luo, X. G. (2010). Optimization of Software components selection for component-based software system development; *Computer & Industrial Engineering*, 58(4), 618-624

Özçevik, Y. (2021). "Simplified QMOOD Model Proposal Based on Correlation Analysis in Different Client Applications," *2021 29th Signal Processing and Communications Applications Conference (SIU)*, Istanbul, Turkey, 2021, pp. 1-4, doi: 10.1109/SIU53274.2021.9477964.

Rajabi & Witt (2020) Rajabi A, Witt C. Self-adjusting evolutionary algorithms for multimodal optimization. Proceedings of the 2020 Genetic and Evolutionary Computation Conference; 2020. pp. 1314–1322.

Rizk-Allah, Hassanien & Slowik (2020) Rizk-Allah RM, Hassanien AE, Slowik A. Multi-objective orthogonal opposition-based crow search algorithm for large-scale multi-objective optimization. *Neural Computing and Applications.* 2020;32(17):13715–13746. doi: 10.1007/s00521-020-04779-w.

Slowik A, Kwasnicka H. (2020). Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications.* 2020;32(16):12363–12379. doi: 10.1007/s00521-020-04832-8.

Wang, Z., Zhang, Q., Li, H., Ishibuchi, H. and Jiao, L. (2017). ON the use of two reference points in decomposition based Multi-objective evolutionary Algorithms; Swarm Evolutionary computing, 34:89-102

Yi JH, Xing LN, Wang GG, Dong J, Vasilakos AV, Alavi AH, Wang L. (2020). Behavior of crossover operators in NSGA-III for large-scale optimization problems. *Information Sciences.* 2020;509(15):470–487. doi: 10.1016/j.ins.2018.10.005.

Zhang, K., Zhang, Y. and Qiujun, H. (2018). Enhancing Comprehensive Learning Particle Swarm Optimization with Local Optima Topology; Information Science, 471(15)
doi:10.1016/j.ins.2018.08.049