Computers, Materials & Continua
DOI:10.32604/cmc.2021.015112
Article

Tech Science Press

Recommender System for Configuration Management Process of Entrepreneurial Software Designing Firms

Muhammad Wajeeh Uz Zaman[1], Yaser Hafeez[1], Shariq Hussain[2], Haris Anwaar[3], Shunkun Yang[4,*], Sadia Ali[1], Aaqif Afzaal Abbasi[2] and Oh-Young Song[5]

[1]University Institute of Information Technology, Pir Mehr Ali Shah Arid Agriculture University, Rawalpindi, 46000, Pakistan
[2]Department of Software Engineering, Foundation University Islamabad, Islamabad, 44000, Pakistan
[3]Department of Electrical, Electronics and Telecommunication Engineering, University of Engineering and Technology, Lahore, 54000, Pakistan
[4]School of Reliability and Systems Engineering, Beihang University, Beijing, 100191, China
[5]Department of Software, Sejong University, Seoul, 05006, South Korea
[*]Corresponding Author: Shunkun Yang. Email: ysk@buaa.edu.cn
Received: 06 November 2020; Accepted: 22 December 2020

Abstract: The rapid growth in software demand incentivizes software development organizations to develop exclusive software for their customers worldwide. This problem is addressed by the software development industry by software product line (SPL) practices that employ feature models. However, optimal feature selection based on user requirements is a challenging task. Thus, there is a requirement to resolve the challenges of software development, to increase satisfaction and maintain high product quality, for massive customer needs within limited resources. In this work, we propose a recommender system for the development team and clients to increase productivity and quality by utilizing historical information and prior experiences of similar developers and clients. The proposed system recommends features with their estimated cost concerning new software requirements, from all over the globe according to similar developers' and clients' needs and preferences. The system guides and facilitates the development team by suggesting a list of features, code snippets, libraries, cheat sheets of programming languages, and coding references from a cloud-based knowledge management repository. Similarly, a list of features is suggested to the client according to their needs and preferences. The experimental results revealed that the proposed recommender system is feasible and effective, providing better recommendations to developers and clients. It provides proper and reasonably well-estimated costs to perform development tasks effectively as well as increase the client's satisfaction level. The results indicate that there is an increase in productivity, performance, and quality of products and a reduction in effort, complexity, and system failure. Therefore, our proposed system facilitates developers and clients during development by providing better recommendations in terms of solutions and anticipated costs. Thus, the increase in productivity and

satisfaction level maximizes the benefits and usability of SPL in the modern era of technology.

Keywords: Feature selection; recommender system; software reuse; configuration management

1 Introduction

Computers have influenced nearly every aspect of our lives and software has now become a need for all types of organizations, ranging from small to medium and large businesses, for use in day-to-day operations. Every business needs to utilize information technology for effective business operations as well as to obtain useful business insights in a time-efficient manner. Software development organizations develop software for all types of businesses. However, every business has its own requirements and needs. Thus, to fulfill the needs of every business, different applications are available, such as Microsoft Office, automated security systems, and various customized features thereof. The demand for these and other software applications (such as websites, games, mobile applications, and desktop software) is extremely high, and the software industry cannot develop all software applications from scratch. Owing to the large-scale production of software applications, changing every software product according to user requirements (mass customization) is a significant challenge for the software development industry. When the number of software applications with similar functionality increases and as those multiple software applications share similar functionalities, they becomes a software product line (SPL) [1]. Recent advancements in the software development field have heightened the need for SPL for mass customization [2]. In SPL, the customization of a family of software is called SPL configuration (SPLC) [3].

SPL engineering (SPLE) is based on a concept of reuse. Software is developed in small reusable code fragments. The code is used to develop different software with similar functionality. This method advocates developing software by reusing existing assets rather than re-inventing the wheel [3,4]. SPL plays an important role in software engineering. The goal of SPLE is to identify and develop similar software with the help of reusable core artifacts that are common in similar software [4]. The SPLE life cycle includes two phases: domain and application engineering [5]. The domain engineering phase enforces the development of reusable artifacts. The application engineering phase enforces the active reuse of reusable artifacts developed in domain engineering to develop software families (or product lines).

SPL products help to meet the massive demand for similar features, but with the benefits of SPL, certain challenges arise, such as customization management (different options or arrangements of requirements in software to facilitate diverse stakeholder perspectives, e.g., Microsoft Office, e-learning, etc.) or configuration management (different arrangements or combinations of software and hardware for system requirements to facilitate varying behaviors of different platforms and diverse perspectives of stakeholders). SPLC management issues include inaccurate selection and prioritization of system configurations during development. Researchers have proposed different criteria for the selection and prioritization of configurations, but this is still a challenging and highly anticipated area within the field of SPL. The issue of configuration/customization preferences is managed by adopting a recommender or recommendation system. Recommender systems (RS) are intelligent bots or software systems that learn from users' preferences and generate personalized recommendations. There are three types of RS: content-based (CB), collaborative filtering-based (CFB), and hybrid [6,7].

CB works by scanning the items' data, the contents, and the internal text of any query performed by users. The words entered by users become keywords for a query on an RS database. The database query returns the results of previously searched or visited items that are similar to the user query. This concept was implemented and introduced by Lops et al. [8]. These types of algorithms can only be applicable when the internal content of any text can be manipulated using natural language processing (NLP) tools. The CFB-type algorithms focus on relevance and average response feedback from users. The user gives feedback for a specific project and obtains a list of similar high-rating features that should be part of a new project. Such algorithms are subdivided into two more categories: matrix factorization (MF) algorithms and k-nearest neighbor collaborative filtering (kNN-CF) algorithms. MF algorithms have been adopted by several researchers [9–11] and showed great results with respect to prediction. These algorithms are significant because of their great results in recommendation systems, which is why they are considered state-of-the-art algorithms. One of the MF algorithms is the biased regularized incremental simultaneous matrix factorization (BRISMF) algorithm, which is in the class of matrix factorization collaborative filtering (MFCF) algorithms [11]. Finally, a hybrid system is a combination of both CB and CFB.

The use of an RS in the development phase would help developers intelligently select and prioritize configurations from the bulk of the reuse configurations. Similarly, clients could easily select an appropriate configuration to ensure high satisfaction. Most studies on SPLC have been carried out with limited focus areas [12–19]. However, far too little attention has been paid to SPLC tools using RS to manage configurations in recent literature, such as feature integrated development environment (feature IDE) [20] and SPL Online Tools (SPLOT) [21]. Because these tools have limitations due to SPL constraints, they only provide partially configured products and do not manage the configuration based on RS. According to a survey of the existing literature, a few studies partially mitigated end-users' and developers' configuration selection and management issues, but these studies were not based on RS during the development of SPL software. The lack of such support affects the time and cost for developers' productivity and can hinder faster application delivery.

Therefore, to address these issues, we propose an RS to manage and recommend configurations. In the context of our research, we adopted a hybrid RS to support software development and to handle clients' requirements. The recommendations are based on historical information and prior developers' experience, provided during configuration development for reuse in new SPL product development. The proposed recommender system (PRS) also facilitates customer customization management. Thus, it provides a platform for customers and developers to select software features along with the estimated software development cost and compensation to developers, respectively. The system provides user-friendly interfaces that help the client in software customization and recommendation of the configuration process. Thus, in the customization management process, the system helps the client in the selection of relevant features and provides feature-specific information along with an opinion of why a given feature is important for the software. It also provides the average estimated cost required to implement the feature. Similarly, for developers, during configuration management based on historical information and prior experience, the system suggests a list of features for the target software, code snippets, libraries, cheat sheets of programming languages and coding references, and also recommends relevant features and configurations. Thus, the PRS not only enables customers to choose the best features according to their customization, but also brings software development teams, working on the

SPL, closer by compensating for development efforts during configuration management. Therefore, the main contributions of our research are as follows:

- In our work, we present a PRS that helps developers to select and manage product configurations and facilitates customers by providing appropriate recommendations for customization of features based on previous user experiences.
- The PRS maintains and utilizes projects' historical data i.e., from requirements to user and developer experiences, of previously completed projects and provides average cost estimates based on the top three relevant projects for developers using the BRISMF algorithm to check SPL configuration.
- To evaluate the performance of the PRS, an experimental study was carried out on three industrial case studies. The results show that the PRS outperforms and increases productivity as compared with other techniques.

Thus, this research provides a guideline for both researchers and practitioners for customization and configuration management, as well as new dimensions for researchers in the SPL domain.

The remainder of this paper is organized as follows. Section 2 describes the theoretical dimensions of the research and examines how previous studies have implemented SPL configuration methods. Section 3 presents the PRS. Section 4 provides the experimental evaluation details of PRS. Results and discussion are presented in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

A significant amount of research has been conducted to derive SPLC methods with the help of RSs. Serious discussions and analyses of SPLC emerged after the 1990s with the proposition and implementation of the feature model. The goal of these representative RSs is to propose different mechanisms to filter out bulky information from large SPLs for improved product line configuration. Some of these RSs provide a prediction-based scenario to predict the best utility of features for users [11–18]. From the literature, few PRSs predict all sets of features that generate better SPLC [18,21–29]. However, the generalizability of much-published research on this issue is limited, and only a few researchers have provided visualization and optimization support.

2.1 Feature Recommender System

Many approaches have attempted to provide a better feature-based RS for SPL, but they have certain limitations. [18] presented a set of heuristics in terms of recommendations that help to arrange several choices and recommend a set of candidate features that must be configured. The limitation of their work is that the authors did not provide proper guidance for choosing from a different set of candidate features. The limitations also indicate that there is a need for an investigation to measure the success rate of the final configuration generated by these recommender heuristics. It is necessary to merge these sets of heuristics for the incremental and cooperative configuration processes. In [14], a model is presented for dynamic decisions aided by guidance to users throughout the process of product configuration. In this approach, two competing features are provided to users, and users must choose from the competing features for their software. This technique helped in ranking better features for future software development, but resulted in ranking inconsistencies. Moreover, the selection process failed to guide users toward better feature selection when both competing features were the same or of no interest to users. The authors in [13,19] presented approaches for decision-makers that use the feature selection and ranking process. Under this approach, all decision-makers that select features to combine should

sit together in a coalition and be provided with a random set of features. They must make a comparative analysis of two randomly selected features and to detect the relevance of both pairs of features to satisfy a particular quality requirement. However, as one feature might satisfy more than one requirement, one feature can be recommended again and again to the user and this flaw of the RS can be distressing for the users. The authors did not provide any statistical analysis in terms of performance, that is, how much faster their approach is in contrast to other approaches in the software configuration process.

2.2 Product Recommender System

Several studies investigating the SPL configuration have suggested tools based on RSs [4,18,25,30,31]. In [17], the authors tailored different data mining interpolation strategies to predict the relevance of a configuration. The authors' approach is primarily based on the voting system. Users rank different features for the configurations of datasets. However, applying this technique in real time hinders the users from voting confidently. The relationships among features lack diversity, such that it may produce biased results. Furthermore, the votes of users are often Idiosyncratic. [16] endorsed a technique named "Invar," which presents to the users a decision-based model with a hard and fast questionnaire and a described set of viable solutions. This is just like a ranking or voting system based on users' votes and feedback. Decision propagation techniques are used to create a product configuration. However, a few indistinct descriptions and even deceptive records might also be delivered within the questionnaires. Moreover, there are no guaranteed proofs or experiments performed to justify that using such questionnaires could develop a more specific end product.

2.3 Configuration Optimization

Much work in recent literature has been performed in this scenario that assists in selecting those features that help in using quality requirements [5,9,22,25,26,32–36]. However, requirements are changing constantly and defining a main function for stakeholders would be very difficult to find relevant features of preference. It is possible that inappropriate and undesired features may be selected for software development in the configuration process.

2.4 Visualization Support

Previous studies reported work that provides visual support for correct product configuration by collecting all quality requirements of features. [37,38] applied feature relations graphs, which are a visual paradigm to visually represent the constraints, impacts, scope of features, and considered features. The objective of this tool is to aid decision-makers with appropriate knowledge to understand the constraints of features to develop an RS for the configuration process. However, this approach was not applied to any configuration process. The results, based on information obtained during this process, might hinder the capabilities of a user to select and approve an appropriate software configuration. There are many other tools such as SPLOT [21], Feature Plugin [36], FeatureIDE [39], and FaMA [17], which have provided improvements to the software configuration to meet industry standards. However, these do not cover all aspects and are not supported properly in the recommendation process. Therefore, to address the issues discussed above, a tool-based RS is proposed that not only provides adequate guidance to users to capture the requirements for their specific software, but also helps developers to rapidly develop software from scratch. Users can select the desired features of the software, which would not only improve the rankings for specific software features, but also help in creating software configurations more

correctly. In addition, developers can manage configurations and artifacts more efficiently. Tab. 1 provides a comparative overview of the abovementioned approaches.

Table 1: Summary of representative approaches

Contribution	Limitation	References
Introduced RS.	No visualization and optimization support.	[22]
Proposed few heuristics to filter features.	No proper guidelines for selecting features.	[18]
Proposed configuration aided models.	Failed to guide towards better feature selection.	[14]
Provides a method for ranking of features.	Did not make any analysis.	[19]
Introduced an enhanced ranking process.	Did not provide evidence about performance of their approach.	[13]
Practiced data mining techniques.	Hinders users to vote confidently.	[30]
Provides decision-based model (Invar)	Chances to get deceptive records.	[16]

3 Proposed Recommender System

In this section, we present the proposed system and describe its internal processes and functionality. Fig. 1 shows the PRS. A client establishes communication with a software development organization by phone calls, websites, email, etc. The company provides access to portable software that is connected with an industry link. This application has three phases. The first phase allows the customer/client to fill in the credentials to create their profiles; these include the client name, company name, type of software that the customer wants to develop, a profile picture, and a few other pieces of information. In the next phase, the customer selects a list of features based on their profile, preferences, and type of software that must be present in the target software.

Many features that are considered as core features are already selected. The system provides a recommendation of potentially core features derived from the top three similar projects completed by the company, including the cost per feature. The system provides an estimated cost based on the average of their previous high-rated features' cost to clients after selecting features for the new product. Similarly, the development team evaluates the development cost of a particular product.

The user interface is dynamic, populating recommended features and removing features that are not supported by the development industry. The client and development team can create a feature checklist based on their requirements. The cost of each selected feature is calculated by the average cost of features from the top three similar completed projects. Clients and developers may select features from different projects or a single project according to the requirements. For a new product development, the system recommends a list of features based on similar previously developed features to reuse, as extracted from the database. These features are reused with their cost for new product feature development instead of rebuilding and recalculating the cost of similar features. Similarly, if more than one previously developed feature is recommended by the system, then we use their costs for the recommendation, on a priority basis, or if they have

similar priority, then we calculate to cost as the average cost of the top three similar features to develop a new feature in the new SPL product. If some features of the product are not available in the top three completed projects, then the feature cost is retrieved from the individual feature list in the database. The features also have constraints that must be considered when generating configurations. Whenever a feature is selected, RS checks and validates the constraints of that feature and generates a configuration.

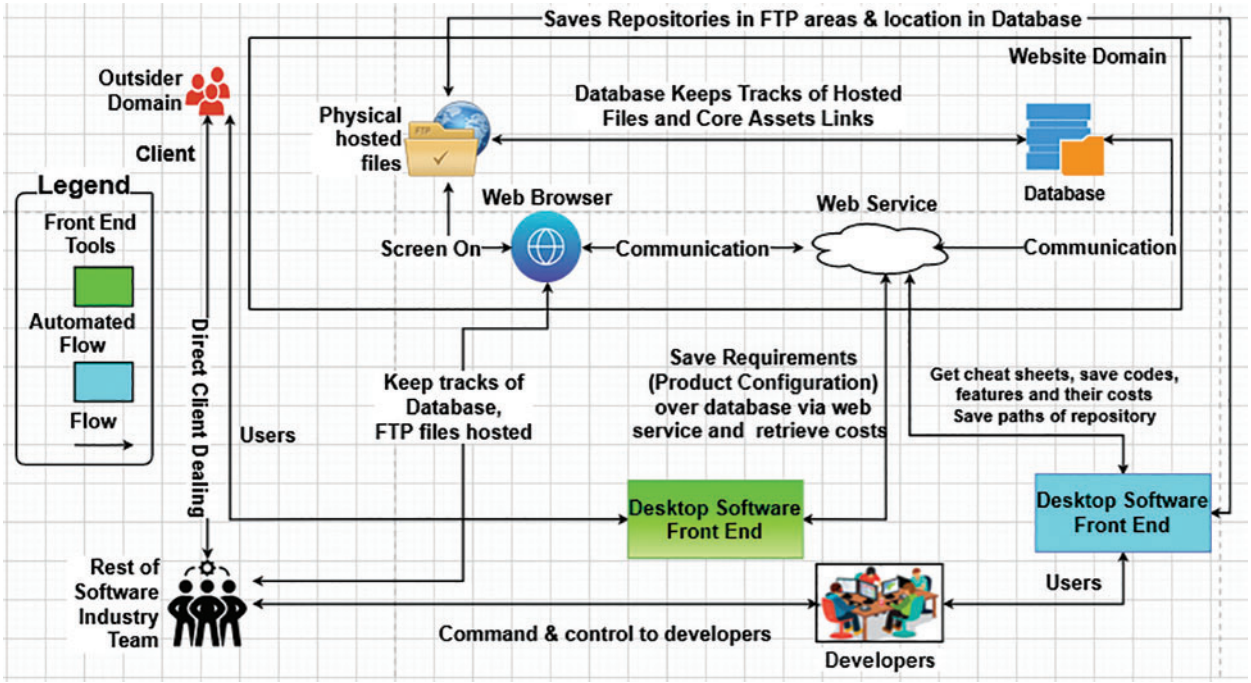

Figure 1: Overview of the proposed recommender system

Four configurations might be collected finally: (1) complete configuration—when features selected by the client fulfill all constraints, (2) partial configuration—when features do not fulfill all constraints, (3) valid configuration—a configuration that satisfies all constraints of features, and (4) invalid configuration—a configuration is considered invalid if it does not satisfy all constraints in the features. At this stage, a sufficient number of configurations can be obtained from the retrieved data, and specific features of the target application can be examined. After the selection of features, a document is produced in the third phase, which serves as an agreement between the client and the company. The client can review and finalize the requirements that will be implemented by the developers. This documentation is either digitally signed or printed and then signed by the client for proof and clarity. This document is then used by developers as a template for development. The system also helps developers to search the repository for code references, host physical files, knowledge about new clients, and to develop as per the client requirements. For the RS, a cloud-based web service is used for both the development team and customer connections with standard interfaces provided by our tool-based support. Fig. 2 shows the internal structure of the cloud-based web service. It contains mechanisms and functions to handle queries from both the client and team and provides corresponding results. Therefore, our PRS consists of two tools: the client tool and developer tool.

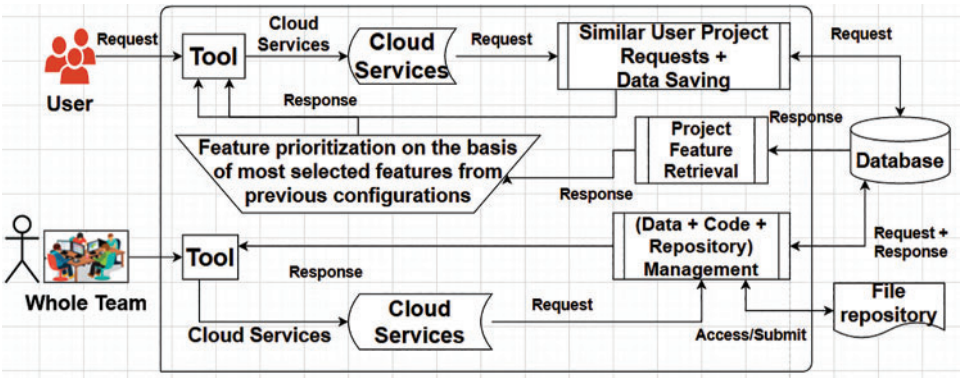

Figure 2: Internal structure of cloud-based web service

3.1 Client Tool

This tool is made available to the clients and provides an interface for capturing the target project's data. First, the client provides demographic information, followed by the details of the target software and its potential features. Whenever a client requests a query and selects a project type, a list of similar projects along with features and costs from the database are suggested to the client, based on features previously selected by similar clients. In addition, features are sorted according to previous clients' ratings after finalizing their products. The RS then creates a model of relevant features and recommends new features using likelihood similarity and neighborhood matching of selected features. Then, along with the client's selected features, the RS also recommends features added from similar software. After analyzing the features, the system generates a list of recommended features along with the objectives and contributions of the project. The feature checklist enables the client to review the features and make a clear selection. The RS tests and validates the constraints every time a feature is selected. Finally, the feature list is stored in the database for use by the development team.

3.2 Developer Tool

The developer tool provides an environment that allows developers to review and analyze the features of the target project, which are listed by the client. Furthermore, it supports developers to check other information about platforms and other features that may be required, and search the repository through the tool's interface. A developer sends the project information to the RS. If a specific user-requested feature is not available, then it will be developed and added to the repository for future reuse. The cost of this developed feature is updated in the database, so that the RS can calculate and provide updated costs to clients.

Fig. 3 illustrates a GUI screenshot that implements cheat sheet support for writing feature code, checking dependencies of other features, adding new features' snippets, searching previous features, and modifying them. This also helps developers to fetch and write a specific code at the cursor position and modify it. A view of recent project repositories is also added to aid developers in extracting code, snippets, and reusable assets from it. The developers' console not only helps in finding and rapidly developing code, but also supports code compilation. Developers use external tools for compilation, and external compilation support has been added to the tool. Support for many popular languages, including C#, C, Java, Python,

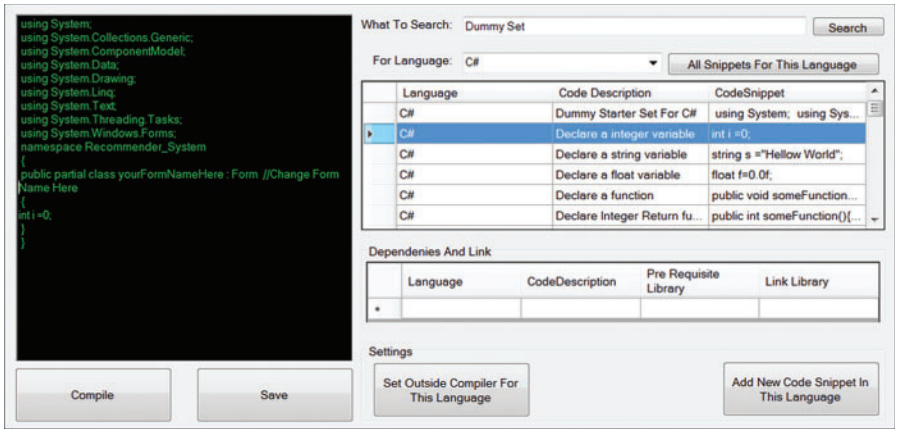

Figure 3: Cheatsheet for feature code

JavaScript, and Ruby has been added. The individual compiler can also be configured according to the requirements. Our PRS is very simple. It takes a dataset to create a model for a recommendation. We recommended similar projects in the client tool and platforms or core assets in the developers' tool. The features and core assets information came from the database, and our PRS created a model from it. Then, from the model, it extracts the likelihood similarity and nearest neighborhood of features and recommends items to the users or developers.

To evaluate the performance of the PRS, the proposed average similarity RS is compared with the neighborhood-based CF RS, matrix RS, and then with open-source unbiased Neo RS. These RSs have been adopted in different studies and practically implemented in the industry as described in the literature [29,38,39]. The neighborhood-based CF recommender provides suggestions based on previous community users' suggestions. Matrix recommenders such as BRISMF use a biased regularized algorithm to factorize a matrix into two low-rank matrices. It takes preprocessed $data(n)$ and $dataset(h)$, and gives $learner(n)$, $predicator(h)$, and latent features of $users(P)$ and $items(Q)$; $Xnxh = Pnxk \cdot Qkxh$, where k is the dimension and P is the Q feature of any Project X required by the client.

4 Experimental Evaluation

In this section, we provide complete details of an experimental study to determine the PRS effectiveness and investigate its performance. For the experimentation and performance evaluation, a local software development organization with experience in SPL product development was selected.

4.1 Datasets

To form a dataset for the PRS, the company's completed projects' data were retrieved and stored in the repository. These data would act as a dataset for the input of the RS to evaluate its performance. The dataset contains the following four properties: (1) the number of features ($\#f$), (2) cross-tree percentage if cross-tree constraints (R), (3) all possible combinations of configurations ($\#c$), and (4) a number of recent configurations ($\#c \rightarrow x$). Features that existed in the application are marked as *yes*, *no* for non-existent, and if not applicable for the respective application, then marked as *notapplicable*. Cross-tree constraints were retrieved based on INNER join in SQL queries. The possible number combinations came from SQL queries in which clauses

were used for the appropriate and filtered results. Recent configurations were retrieved using simple and less strict SQL queries. Tab. 2 shows the Chat applications created by the company and their relevant parameters. Tab. 3 lists the games developed based on Unity games, associated packages purchased, and their relevant parameters. Tab. 4 shows EE-based completed projects. Based on the company's profile and development history that shows that the company has already completed 578 projects in various domains, it would be beneficial for users to conduct case studies in the selected company.

Table 2: Similar chat software produced by the company with the first clients' requirements

Projects	Application type	#f	R (%)	#c	#c → x
Simple chat application	Networking	1290	4	$>10^9$	341
Chat application with encryption decryption	Networking	1354	51	$>10^9$	548
Wajeeha global chat application	Networking	1150	9	$>10^9$	271

Table 3: Similar unity game projects owned by the company

Projects	Application type	#f	R (%)	#c	#c → x
Intelligent Traffic System (iRTS)	Unity project	432	65	$>10^9$	453
Ady's physics vehicles	Unity project	124	45	$>10^9$	548
Low poly car models with controllers	Unity asset	103	53	$>10^9$	234

Table 4: Similar website projects done by the company

Projects	Application type	#f	R (%)	#c	#c– > x
Online car selling project	E-commerce website	12	6	$>10^9$	118
Bidding project	E-commerce website	14	9	$>10^9$	120
ONeX buy and sell	E-commerce website	17	12	$>10^9$	151

Developers' code compensation is monitored by WireShark, a network analyzer tool, and with Spy code in the software application. Wireshark shows and captures all websites visited by a network of computers. The Spy code is used in the developer console to monitor the queries made by users for a specific language and code, and to track any new code snippet added during the development. Wireshark is used to analyze and compare network data.

4.2 Experimental Design

The input data for the algorithm must be tuned before it is processed. If the parameters are tuned manually, the algorithm may yield biased results that may influence the research validity. To ensure unbiased tuning, we used a genetic algorithm to set the inputs for the average, KNN-CF, and BRISMF. The constructed dataset was used for training and performance evaluation of the representative recommendation algorithms. The value of the F-measure is optimized for all three

algorithms with the help of a genetic algorithm. To further evaluate the practical assessment, we answer the following research questions (RQs) to show that the PRS improves stakeholders' and customers' performance, project quality, and achieves more accurate predictions than other RSs.

RQ1: Does our recommender system help in SPL configuration?

Ans: Yes, it generates a list of recommended features based on three recently completed software applications, developed in the same category by the company. Thus, the client can select features with ease and less effort.

RQ2: How is SPL configured with our approach?

Ans: Features are retrieved from the repository datasets and saved according to the client's requirements. Developers fetch required features, find them in the repository; if not found, they develop and store features in repositories, combine features, and deploy them to the client.

RQ3: How accurately are feature costs calculated by our RS?

Ans: Costs are saved in the repository at the time of feature development, retrieved, and included when a client selects features for its required software. The cost may be increased or decreased for a specific feature depending on economic factors, taxes, and other industrial factors. As in SPL, some features are common and available in all product series with a fixed cost. Subsequently, certain features are variables and have different costs. To control the variable feature cost calculation, our system provides a list of features from the database, which is similar to new variable features for reusing during new SPL development. These features may be previously developed or purchased for product development in similar domains. The list of features helps the developer manage configuration and cost by selecting appropriate and lower-cost features for variable configurations.

RQ4: How does the proposed system help during development?

Ans: It provides aid to developers to quickly search for code in the repository or add/update where required. It has the ability to compile code using the compiler chosen by the developer during the software configuration. Developers can add more code snippets that can be made accessible to other developers in the same company network, and download files, assets, etc., from the repository for practical reuse.

The analysis of the results shows that the PRS improves the configuration management process of SPL development and guides both developers (for configuration selection and prioritization during reuse of features) and customers (for customization management to use accurate features according to needs) by recommending appropriate information.

4.3 Training Database Datasets

To simulate a realistic environment for the evaluation of certain conditions, a PRS of incremental data configuration retrieval is used in this work. According to the PRS, one configuration is fetched from the set of configurations, and the selected configuration is already used for analysis. The persisting set of configurations acts as training data for the algorithm. The PRS advocates a real-time system in which a new configuration is made when any new user logs in. The information of past users' configurations is available as a dataset or training data, through which the prediction of a newly created configuration would be performed. In order to generalize the best results by an RS, that system must recommend those features that were used in the excluded configuration test. This configuration is based on all the other configurations of the training data.

4.4 Evaluation Metrics

To evaluate our work, F-measure, precision, and recall methods at "w" were used, where "w" is the set of recommendations allowed for testing and its size was three in our evaluation method. *Rec* is the set of features that are recommended by all four algorithms and used the PRS for evaluation. *Rel* is the set of features that are truly relevant to the test configuration. The precision (P) can be calculated as $P = \frac{|Rec \cap Rel|}{\omega}$. Therefore, the recall (R) is calculated as $R = \frac{|Rec \cap Rel|}{|Rel|}$. The F-measure is calculated as $F - Measure = \frac{2*P*R}{P+R}$.

5 Results and Discussions

In this section, we discuss the results of the experimental study to demonstrate the effectiveness of the PRS. For a comparative analysis of the results, we used "Neo Rec." This tool's application programming interface (API) was integrated with our tool in such a way that it fetches a random list of unselected recommendation features. It is very critical to use the Neo Rec system, as it provides a real and optimistic level of performance that all types of recommenders should reach. If the test fails and the PRS does not outclass this random API-based recommender, then there would be no value of our contribution, and using it should not outperform the results. Moreover, the overall performance of this tool is equal to the performance of a hypothetical uninformed user without any support from an RS.

Fig. 4 (left) and (right) show screenshots of the client's interaction and target project information respectively. The selected company was working on three software applications for different clients from Saudi Arabia. Fig. 5 presents a screenshot of Perform Menu that helps developers to navigate to the desired operation that they want to perform. Fig. 5 illustrates the interface through which developers can retrieve client requirements.

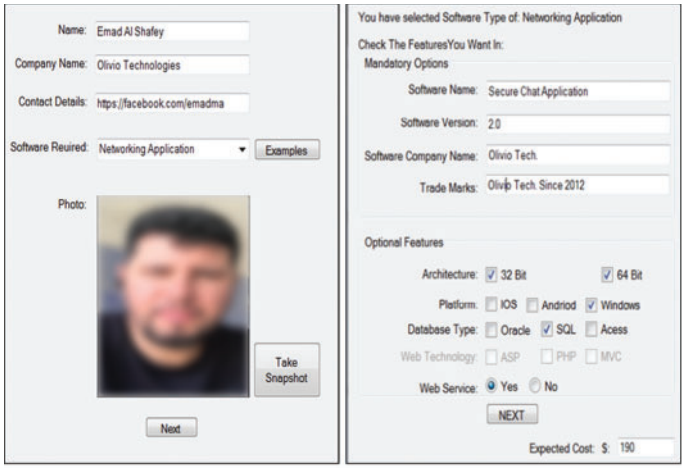

Figure 4: First and second views of the client application

The performance analysis of each representative RS in terms of similarity measure is presented. The F-measure for each representative RS for three types of applications, i.e., chat, gaming, and web-based is calculated and shown in Figs. 6–8, respectively. The results demonstrate that the proposed recommender system outperforms other RSs, as it creates more datasets of configurations from training data. There is a slightly significant difference between the CF and

BRISMF algorithms, as their bars in the abovementioned figures overlapped. The results shown in Figs. 6–8 indicate that both CF and BRISMF dominated the random recommender throughout all phases of the configuration process apart from the starting stage, where limited information was available in the dataset regarding configuration. Further analysis shows the dominance of the average similarity algorithm not only for the F-measure but also in terms of precision and recall of configurations. Figs. 9–11 provide the results of the F-measure, precision, and recall of the three case study datasets of chat, website, and unity applications, respectively.

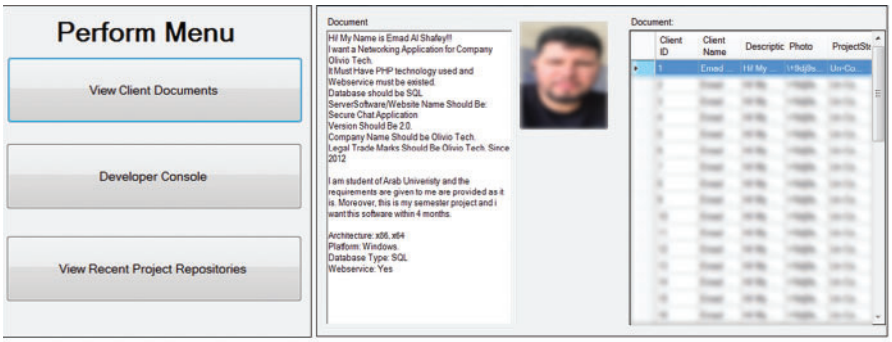

Figure 5: Developers' perform menu and screenshot of client requirements

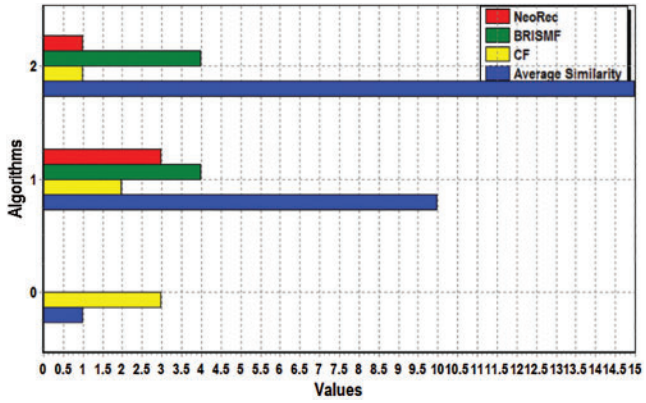

Figure 6: F-measure achieved by four different recommendation algorithms for a chat application

From the results, it can be seen that our proposed average similarity RS successfully identified and recommended correct features to the client. Owing to the privacy policy of the company, developers' browser data and code snippets added are not disclosed. The code snippets contain confidential credentials and business process information and are therefore restricted. However, the company allowed us to share numerical data concerning analytics. In two months, 453 code queries, bugs, and exceptions were searched, and 254 code snippets were added to seven different software types. Customer feedback is essential to assess customer satisfaction with a product or service. Hence, the customer was asked to rank the delivered projects using star ratings. The customer assigned 4-, 4-, and 5-star ratings to the chat application, unity game, and website, respectively. Fortunately, the company used the Agile Scrum Method for software development, and it delivered software to the client within two sprints (i.e., two months).

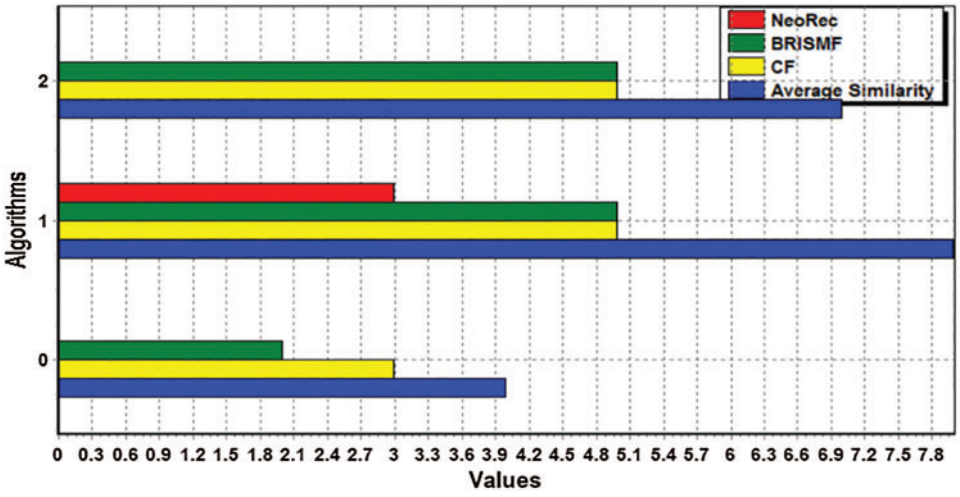

Figure 7: F-measure achieved by four different recommendation algorithms for the unity game project

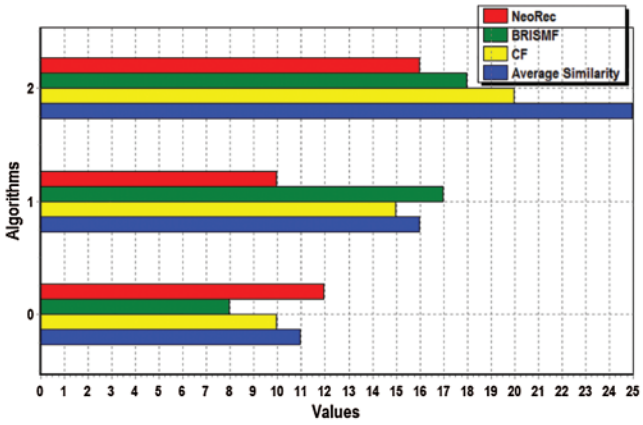

Figure 8: F-measure achieved by four different recommendation algorithms for the website project

Four types of validity threats have been analyzed from the existing literature [40–42]. The internal validity threat is the arrangement of all requirements to determine the effectiveness of the PRS for problem solving. Thus, our PRS deals with two different types of views (i.e., client and developer) of problems of selecting appropriate options, and we conducted an industrial experiment for evaluating the PRS. The results are compared with other existing RSs and reveal that the PRS improves feature selection and manages development cost. To reduce external validity threats, we selected a real dataset of industrial applications for an accurate evaluation of the PRS as compared to dummy data or projects. This increases the efficiency of our system and can be applied to similar projects to reproduce the same or similar results. The work also provides guidelines to improve the recommendation problems of both clients and developers. Therefore, our system is useful for managing different perspectives of clients and developers for feature selection using different datasets of projects during the experiment.

In construct validity, the verification and validation of the PRS relationship among the theory and findings analyzed. Thus, we analyzed relevant literature to find evidence supporting the need for the PRS with an empirical evaluation using industrial case studies. Subsequently, validity threats with respect to unbiased and rigorous experiment results were determined. Therefore, for reliable results, all authors reviewed and analyzed all the results and their descriptions using statistical analysis. This ensured that the results on different datasets are similar or may diverge slightly within the same conditions, rather than yielding completely different results.

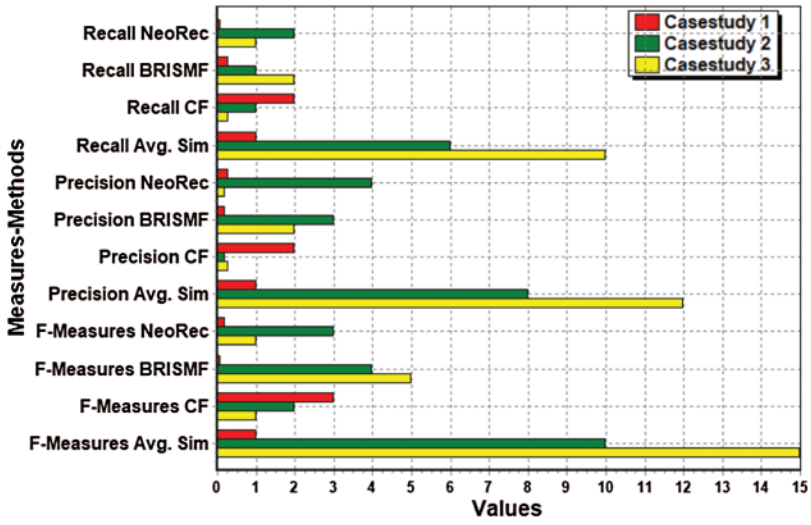

Figure 9: Recommender systems performance for a chat application

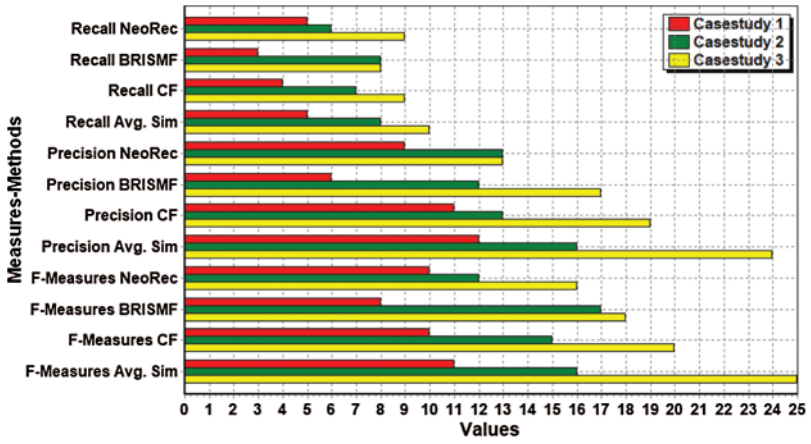

Figure 10: Recommender systems performance for the website

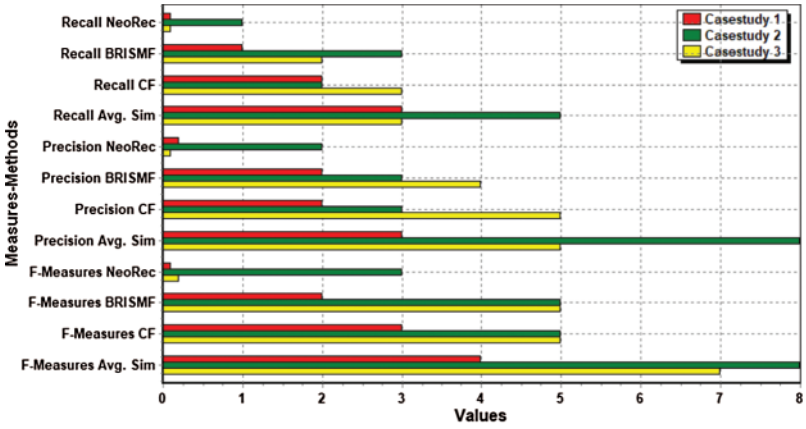

Figure 11: Recommender systems performance for the unity game

6 Conclusion and Future Work

This study presents an RS to resolve the issues of development costs, and feature recommendations of a configuration management system during software development, which are faced by developers as well as customers for the correct selection of appropriate features. In this work, we propose an RS, a desktop application that helps customers and developers to choose the type of software to be developed and selects its features. By utilizing the data of earlier completed projects within a similar domain, the system generates a list of recommended features for the target project as well as a cost estimation. This allows the customer to choose features based on their importance and dependability, and their overall role within the project's scope. The user-friendly interfaces of the proposed system provide a self-descriptive roadmap that guides the customer to fetch all requirements and features conveniently and quickly. Similarly, for developers, during configuration management based on historical information and prior experience, the system suggests a list of features for the target software, code snippets, libraries, cheat sheets of programming languages and coding references, and also recommends relevant features and configurations. The empirical results show that developers feel compensated during the development effort. This increases the overall productivity of the company, quality of the product being developed, meeting budget constraints, and increasing customer satisfaction and experience. For future work, we plan to extend the domain of component-based software development and will attempt to increase the reusability of features, to manage changes and decision-making activities of the development team.

Funding Statement: The work reported in this manuscript was supported by the National Natural Science Foundation of China (Grant Number: 61672080, Sponsored Authors: Yang S., Sponsors' Websites: http://www.nsfc.gov.cn/english/site_1/index.html).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] K. Pohl, B. Günter and F. J. van Der Linden, "Software product line engineering: Foundations, principles and techniques. Berlin Heidelberg: Springer Science & Business Media. Verlag, 2005. [Online]. Available: https://link.springer.com/book/10.1007/3-540-28901-1.

[2] F. J. van Der Linden, K. Schmid and E. Rommes, "Software product lines in action: The best industrial practice in product line engineering. Berlin Heidelberg: Springer Science & Business Media. Verlag, 2007. [Online]. Available: https://link.springer.com/book/10.1007/978-3-540-71437-8?page=1.

[3] F. Ahmed and L. F. Capretz, "The software product line architecture: An empirical investigation of key process activities," *Information and Software Technology*, vol. 50, no. 11, pp. 1098–1113, 2008.

[4] J. A. Pereira, S. Schulze, S. Krieter, M. Ribeiro and G. Saake, "A context-aware recommender system for extended software product line configurations," in *Proc. of the 12th Int. Workshop on Variability Modelling of Software-Intensive Systems*, Madrid, Spain, pp. 97–104, 2018.

[5] M. Asadi, E. Bagheri, B. Mohabbati and D. Gašević, "Requirements engineering in feature oriented software product lines: An initial analytical study," in *Proc. of the 16th Int. Software Product Line Conf.*, Salvador, Brazil, vol. 2, pp. 36–44, 2012.

[6] O. Palombi, F. Jouanot, N. Nziengam, B. Omidvar-Tehrani and M. Rousset, "OntoSIDES: Ontology-based student progress monitoring on the national evaluation system of French Medical Schools," *Artificial Intelligence in Medicine*, vol. 96, pp. 59–67, 2019.

[7] M. Elazony, A. Khalifa, S. Nouh and M. Hussein, "Design and implementation of adaptive recommendation system," *International Journal of Management, Technology, and Social Sciences*, vol. 3, no. 1, pp. 101–117, 2018.

[8] P. Lops, M. de Gemmis and G. Semeraro, "Content-based recommender systems: state of the art and trends," in *Recommender Systems Handbook*. Boston, MA: Springer, pp. 73–105, 2011. [Online]. Available: https://link.springer.com/chapter/10.1007/978-0-387-85820-3_3.

[9] Y. Koren, "Collaborative filtering with temporal dynamics," in *Proc. of the ACM Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, Paris, France, pp. 447–456, 2009.

[10] Y. Koren, R. Bell and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[11] G. Takács, I. Pilászy, B. Németh and D. Tikk, "Scalable collaborative filtering approaches for large recommender systems," *Journal of Machine Learning Research*, vol. 10, pp. 623–656, 2009.

[12] E. Bagheri, T. D. Noia, A. Ragone and D. Gasevic, "Configuring software product line feature models based on stakeholders," in *Soft and Hard Requirements, in Proc. of Int. Conf. on Software Product Lines*, Jeju, South Korea, pp. 16–31, 2010.

[13] E. Bagheri, M. Asadi, D. Gasevic and S. Soltani, "Stratified analytic hierarchy process: Prioritization and selection of software features," in *Proc. of Int. Conf. on Software Product Lines*, Jeju, South Korea, pp. 300–315, 2010.

[14] E. Bagheri and F. Ensan, "Dynamic decision models for staged software product line configuration," *Requirements Engineering*, vol. 19, no. 2, pp. 187–212, 2014.

[15] E. Bagheri, T. D. Noia, D. Gasevic and A. Ragone, "Formalizing interactive staged feature model configuration," *Journal of Software: Evolution and Process*, vol. 24, no. 4, pp. 375–400, 2012.

[16] J. A. Galindo, D. Dhungana, R. Rabiser, D. Benavides, G. Botterweck *et al.,* "Supporting distributed product configuration by integrating heterogeneous variability modeling approaches," *Information and Software Technology*, vol. 62, pp. 78–100, 2015.

[17] J. Martinez, G. Rossi, T. Ziadi, T. F. D. A. Bissyandé, J. Klein *et al.,* "Estimating and predicting average likability on computer-generated artwork variants," in *Proc. of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, Madrid, Spain, pp. 1431–1432, 2015.

[18] R. Mazo, C. Dumitrescu, C. Salinesi and D. Diaz, "Recommendation heuristics for improving product line configuration processes," in *Recommendation Systems in Software Engineering*. Berlin, Heidelberg: Springer, pp. 511–537, 2014, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-45135-5_19.

[19] L. Tan, Y. Lin and L. Liu, "Quality ranking of features in software product line engineering," in *Proc. of the Asia-Pacific Software Engineering Conf.*, Jeju, South Korea, vol. 2, pp. 57–62, 2014.

[20] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake *et al.,* "FeatureIDE: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, 2014.

[21] M. Mendonça, M. Branco and D. Cowan, "S.P.L.O.T.: Software product lines online tools," in *Proc. of the Conf. on Object-Oriented Programming, Systems, Languages and Applications*, Orlando, Florida, USA, pp. 761–762, 2009.

[22] A. Felfernig, S. Polat-Erdeniz, C. Uran, S. Reiterer and M. Atas, "An overview of recommender systems in the internet of things," *Journal of Intelligent Information Systems*, vol. 52, no. 2, pp. 285–309, 2019.

[23] E. D. Farahani and J. Habibi, "Configuration management model in evolutionary software product line," *International Journal of Software Engineering and Knowledge Engineering*, vol. 26, no. 3, pp. 433–455, 2016.

[24] J. Bosch, R. Capilla and R. Hilliard, "Trends in systems and software variability," *IEEE Software*, vol. 32, no. 3, pp. 44–51, 2015.

[25] J. Rodas-Silva, J. A. Galindo, J. Garcia-Gutierrez and D. Benavides, "Selection of software product line implementation components using recommender systems: An application to Wordpress," *IEEE Access*, vol. 7, pp. 69226–69245, 2019.

[26] T. Kim, S. Kang and J. Lee, "Effects of variable part auto configuration and management for software product line," in *Proc. of 2018 IEEE 42nd Annual Computer Software and Applications Conf.*, Tokyo, Japan, vol.1, pp. 827–828, 2018.

[27] J. W. Payne, J. R. Bettman and E. J. Johnson, *The Adaptive Decision Maker*. New York, USA: Cambridge University Press, 1993.

[28] J. A. Pereira, K. Constantino and E. Figueiredo, "A systematic literature review of software product line management tools," in *Proc. of the Int. Conf. on Software Reuse*, Miami, FL, USA, pp. 73–89, 2015.

[29] J. A. Pereira, C. Souza, E. Figueiredo, R. Abilio, G. Vale *et al.,* "Software variability management: An exploratory study with two feature modeling tools," in *Proc. of the Brazilian Symp. on Software Components, Architectures and Reuse*, Brasilia, Brazil, pp. 20–29, 2013.

[30] J. Cleland-Huang and B. Mobasher, "Using data mining and recommender systems to scale up the requirements process," in *Proc. of the 2nd Int. Workshop On Ultra-large-scale Software-Intensive Systems*, Leipzig, Germany, pp. 3–6, 2008.

[31] J. A. Pereira, P. Matuszyk, S. Krieter, M. Spiliopoulou and G. Saake, "A feature-based personalized recommender system for product-line configuration," in *Proc. of the 2016 ACM SIGPLAN Int. Conf. on Generative Programming: Concepts and Experiences*, Amsterdam, Netherlands, pp. 120–131, 2016.

[32] R. M. Hierons, M. Li, X. Liu, S. Segura, W. Zheng *et al.,* "Optimal product selection from feature models using many-objective evolutionary optimization," *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 2, pp. 1–39, 2016.

[33] X. Lian and L. Zhang, "Optimized feature selection towards functional and non-functional requirements in software product lines," in *Proc. of the Int. Conf. on Software Analysis, Evolution, and Reengineering*, Montreal, QC, Canada, pp. 191–200, 2015.

[34] L. Machado, J. Pereira, L. Garcia and E. Figueiredo, "SPLConfig: Product configuration in software product line," in *Proc. of the Brazilian Conf. on Software: Theory and Practice*, Alagoas, Brazil, pp. 1–8, 2014.

[35] G. G. Pascual, R. E. Lopez-Herrejon, M. Pinto, L. Fuentes and A. Egyed, "Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications," *Journal of Systems and Software*, vol. 103, pp. 392–411, 2015.

[36] T. H. Tan, Y. Xue, M. Chen, J. Sun, Y. Liu *et al.,* "Optimizing selection of competing features via feedback directed evolutionary algorithms," in *Proc. of the Int. Symp. on Software Testing and Analysis*, Baltimore, MD, USA, pp. 246–256, 2015.

[37] H. Wu and G. Li, "Visual communication design elements of internet of things based on cloud computing applied in graffiti art schema," *Soft Computing*, vol. 24, no. 11, pp. 8077–8086, 2020.

[38] E. D. Farahani and J. Habibi, "Feature model configuration based on two-layer modelling in software product lines," *International Journal of Electrical & Computer Engineering*, vol. 9, no. 4, pp. 2648–2658, 2019.

[39] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature modeling plug-in for eclipse," in *Proc. of the Conf. on Object–Oriented Programming, Systems, Languages and Applications Workshop on Eclipse Technology Exchange*, Vancouver, British Columbia, Canada, pp. 67–72, 2004.

[40] M. Felderer and A. Herrmann, "Comprehensibility of system models during test design: A controlled experiment comparing UML activity diagrams and state machines," *Software Quality Journal*, vol. 27, no. 1, pp. 125–147, 2019.

[41] B. Miranda and A. Bertolino, "An assessment of operational coverage as both an adequacy and a selection criterion for operational profile based testing," *Software Quality Journal*, vol. 26, no. 4, pp. 1571–1594, 2018.

[42] J. F. S. Ouriques, E. G. Cartaxo and P. D. Machado, "Test case prioritization techniques for model-based testing: A replicated study," *Software Quality Journal*, vol. 26, no. 4, pp. 1451–1482, 2018.