A Non-Functional Requirements Recommendation System for Scrum-based Projects

Felipe Ramos[§], Alexandre Costa[§], Mirko Perkusich[¶], Hyggo Almeida[§] and Angelo Perkusich[§]
[§]Intelligent Software Engineering (ISE) Group, Federal University of Campina Grande,
Campina Grande, Paraíba, Brazil, 58429-140
{feliperamos, antonioalexandre}@copin.ufcg.edu.br,
hyggo@dsc.ufcg.edu.br, perkusic@dee.ufcg.edu.br
[¶]Federal Institute of Paraíba, Monteiro, Paraíba, Brazil, 58500-000
mirko.perkusich@ifpb.edu.br

***Abstract*—Agile software development focuses on quick delivery and flexibility to change. Despite being effective in delivering quality functional requirements, agile practices tend to neglect non-functional requirements until the later stages of software development. This work focuses on Scrum, the most popular agile method, and presents a non-functional requirements recommendation system to support Scrum practitioners on their early identification. The solution is based on instrumenting the Scrum process to extract useful data and the use of collaborative filtering and item recommendation. To evaluate the recommendations, we conducted off-line experiments with data collected from 12 Scrum practitioners through a survey. The data was analyzed using 10-fold cross-validation. As a result, our proposed solution showed a recall rate of up to 81%, which indicates that it is a promising approach to recommend non-functional requirements given a set of functional requirements identified by project stakeholders.**

***Keywords*—Non-functional Requirements; Scrum; Recommendation System; Agile Software Development.**

I. Introduction

Agile software development (ASD) methods, such as Scrum and XP, have gained strength with the acceptance of the fact that uncertainty is part of software development [4], emerging as an alternative to keep up with the high competitiveness and volatility of the software market. Unlike traditional approaches, which rely on detailed processes and extensive planning, ASD focuses on the rapid delivery of business value to customers. Moreover, ASD supports requirement changes at any stage of the development process [4]. However, as well as in traditional approaches, Requirements Engineering (RE) is a crucial process for the success of agile projects.

ASD methods, in contrast to traditional plan-driven processes, follow an incremental and iterative development process. Even though they are efficient in delivering quality functional requirements (FRs) [7], non-functional requirements (NFRs) are often overlooked until the later stages of software development [10], which might increase the costs [10] and probability of failure [2].

This work focuses on Scrum, the most popular agile method. In Scrum, the requirements are usually managed by a person with a business-oriented profile (i.e., the Product Owner), which tends to focus on FR, neglecting NFRs. [13].

Most existing studies related to applying NFRs processes to ASD [10], [6], [5] do not consider Scrum's artifacts, events and roles, reducing their applicability. The studies [14], [13], [3] that focus on Scrum present processes to complement it and consider, for instance, modeling NFRs as a "done" criteria or a constraint story [13].

Our goal is to automate the definition of NFRs given historical data of Scrum projects, supporting the team on identifying them early in the process. For this purpose, in this paper, we present a NFRs recommendation system to support Scrum practitioners. It is based on collaborative filtering and item recommendation and supported by an instrumentation of the Scrum process to enable the collection of quality data for the recommendations.

This paper is organized as follows. Section II presents previous works on NFRs applied to the ASD. Section III presents the proposed solution. Section IV presents the design of the validation process. Section V discusses the results; and Section VII presents our conclusions and future work.

II. Related Work

There are several studies regarding the management of NFRs on ASD [3], [5], [6], [7], [8], [10], [11], [13], [14], [15].

In a study including a series of papers, authors presented solutions for the capture, definition and prioritization of NFRs in ASD. First, authors proposed a NFR modeling framework that is tailored for agile processes, called Non-Functional Requirements Modeling for Agile Processes (NORMAP) [6]. In addition, a simulation tool was proposed for modeling non-functional requirements for semi-automatic agile processes (NORMATIC) [7], which supports the more general NORMAP methodology. In [5], authors proposed a methodology for elicitation, reasoning and validation of NFRs in agile processes (NERV), which showed better results in comparison to the NORMAP framework. NERV is a lightweight methodology to address NFRs early in ASD. In the study [10], the authors proposed to use NFRs metadata from software requirements artifacts - documents and images - as an extension of previous works [6] and [5]. Finally, in [12], authors investigated the prioritization of requirements based on the framework proposed in [11].

Some studies conducted research on the topic of NFRs in the context of Scrum [3], [13], [14].

DOI reference number: 10.18293/SEKE2018-107

Bourimi et al. [3] proposed the Agile Framework For Integrating Non-functional requirements Engineering (AFFINE) with the goals of: (1) conceptually considering NFRs early in the development process, (2) explicitly balancing end-users' with developers' needs, and (3) having a reference architecture to support NFRs. The authors introduced the role of an NFR stakeholder. Although the proposed solution presents contributions, the method is based only on a conceptual effort of the early consideration of NFRs.

On the other hand, Sabry and El-Rabbat [13] discussed about architectural refactoring framework and techniques for achieving required levels of NFRs through the formalization of Spikes and Definition of Done (DoD) within Scrum practices.

Sachdeva and Chung [14] proposed a novel approach to handle non-functional requirements of security and performance in Scrum-based projects involving big data and Internet of Things (IoT). In their approach, authors proposed to consider security as system functionalities (set of user stories) and performance as spikes and acceptance criteria of user stories.

Although previous studies focused on the early definition of NFRs in ASD, they based their approaches on conceptual reinforcement or on the automatic capture of NFRs from project documents, which may not always be available at the beginning of agile projects. In this study, we focus on the early definition of NFRs, but unlike previous works, we propose the use of historical (considering Scrum roles, artifacts and events) to generate recommendations of NFRs.

III. Proposed Solution

In this section, we present the NFR recommendation system, which aims to support Scrum practitioners in the early definition of NFRs.

In Figure 1, we present an overview of the recommendation process, which is based on the Scrum instrumentation, presented as follows. The activities take place during Sprint Planning meetings: (1) the Scrum Team, with the support of the Scrum Architect (SA), details product backlog items using Semi-structured User Stories (SUS) (2). The SUSs is used by the recommendation system (3) to generate recommendations of NFRs for each FR; (4) The Product Owner (PO) and the Development Team evaluate the recommendations, accepting or rejecting them. If recommendations are accepted, the information about recommended NFRs is stored. If they are rejected, negative feedback is stored.

A. Scrum Instrumentation

We instrument the Scrum framework by adding two new elements, as shown in Figure 2. We added a new role, the **Scrum Architect** (see Figure 2 (1)), and a new artifact based in tags and categories to represent product backlog items: **Semi-structured User Story** (see Figure 2 (3)). We detail the new role and artifact as follows.

1) Semi-structured User Story

In Scrum, user stories are generally used to represent product backlog items. They are composed of three aspects: written description, conversations about their details and a list of acceptance criteria. They usually have the following format:

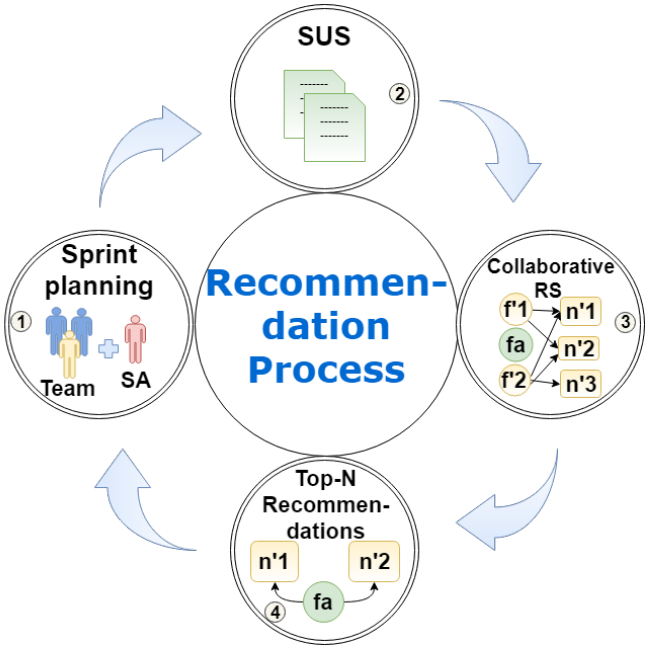

Fig. 1: An overview of the recommendation process.

As a [*type of user*], I want [*some goal*] so that [*some reason*].

This level of information is not enough to retrieve useful data for the recommendation system. Therefore, to ensure the traceability of requirements among different projects, we propose the SUS (see Figure 3).

The SUSs are developed during Sprint Planning meetings and, compared to traditional user story, they contain the following additional information:

- **FR category:** represents a predefined category that classifies a user story based on its goal (high-level FR). For example, "Login", "Alarm and Notifications", "Report Visualization", etc.;
- **Technology tags:** represent labels related to the technologies necessary for the development of a user story. For example, programming language (e.g., java, python, etc.), database (e.g., mongo, sql, etc.), etc.;
- **Associated NFRs:** set of quality attributes that are associated with the functional requirement represented by the user story. Each NFR presents the following information:
  - **NFR category:** represents a predefined category, which classifies the non-functional requirement (e.g., security, performance, privacy, etc.);
  - **NFR statement:** represents the condition of the NFR that must be met to consider the associated functional requirement done. For example, "SSL encryption" for a given FR.

In Figure 3, we present an example of a SUS, in which FR category is "Login" (1), technology tags are "android", "java" and "mongo" (2), and presents an associated NFR (3) of "Security" (4) with "SSL encryption" as statement (5).

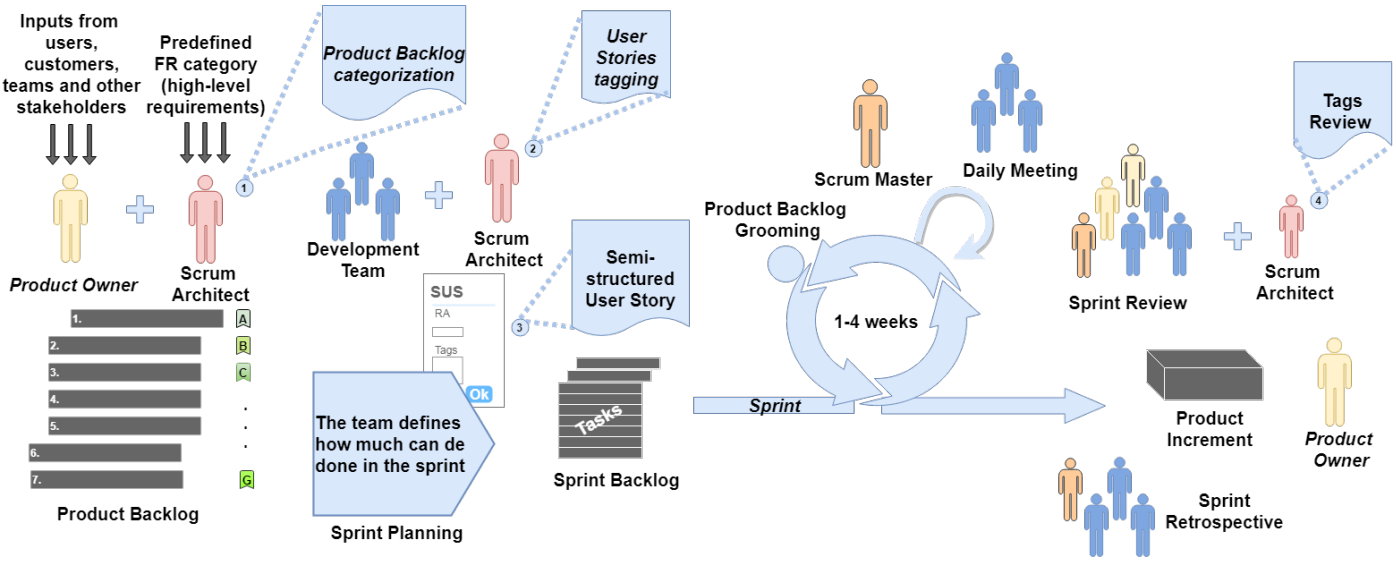

Fig. 2: An overview of the instrumented Scrum process.

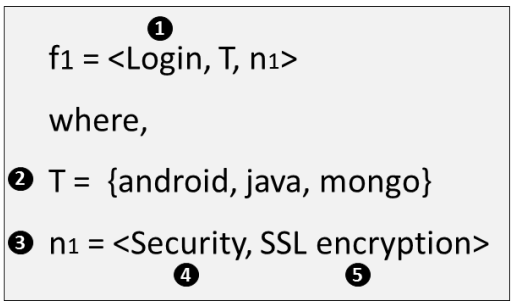

Fig. 3: An example of a Semi-structured User Story.

2) Scrum Architect

The SA is responsible for the following three activities: i) categorize product backlog items and gather project information (see Figure 2 (1)); ii) assign tags to the SUSs (see Figure 2 (2)); and (iii) review previously assigned tags (see Figure 2 (4)). The activities are executed at the beginning of the project, during the initial definition of the product backlog and, continuously, in the Scrum Planning and Review meetings at each Sprint.

During the initial definition of the project backlog, the SA is responsible to assign a predefined category for each product backlog item defined by the PO. The decision on the category of each item should be defined in common agreement between PO and SA by face-to-face communication. If an specific requirement can not be classified in any of the predefined categories, the SA can create a new category, insert it into the dataset, and reuse it (i.e., reuse-driven approach). By categorizing product backlog items, we aim to classify requirements of same purpose, and later, to ease the information retrieval. Additionally, the SA is responsible for filling the project profile with information such as project category (e.g., web, mobile, embedded, desktop, etc.), project domain (e.g., health, banking, etc), project goal (e.g., product or prototype), project architecture (e.g., client-server, MVC, multilayered, etc.), and assign technology tags to the project, which represent the basic technologies needed in the software product development (e.g., programming language, database, etc.).

During Sprint Planning meetings, the SA is responsible for obtaining the remaining information of the SUSs by following the discussions between the PO and the Development Team, ensuring that all product backlog items are categorized and technology tags are properly assigned. Since it is during the Sprint Planning meeting that the Development Team defines all the tasks that must be performed to complete each selected user story for a Sprint, the SA has the chance to accurately obtain the information on which technologies will actually be used to complete each one of them, based on direct feedback from the Development Team.

Finally, during the Sprint Review meeting, the SA is responsible for reviewing the tags assigned to user stories, i.e., to identify if any tag should be removed from a user story representation and/or if new tags should be assigned to it.

B. NFR Recommendation

We address the definition of NFRs as a recommendation problem. Thus, by adapting the generic definition of recommendation problems presented by Adomavicius and Tuzhilin [1], we have the following: let F be the set of all FRs and let N be the set of all possible NFRs that can be recommended. Let u be an utility function that measures the usefulness of a NFR n to a FR f, i.e., u: $F \times N \rightarrow T$, where T is a totally ordered set. Then, for each FR $f \in F$, we want to choose such NFR $n' \in N$ that maximizes the utility of the FR f. More formally:

$$\forall f \in F, \quad n'_f = \arg\max_{n \in N} u(f, n). \tag{1}$$

As follows, we present the main components of the NFR recommendation system:

- **Data Collector:** collects information from SUSs and project profiles to generate FR and NFR profiles;
- **Profile Manager:** generates FRs and NFRs profiles based on information collected by the Data Collector;
- **Recommender:** analyzes FRs and NFRs profiles to generate customized NFRs recommendations.

We detail each one of the three components in the following subsections.

1) Data Collector

The data collection task consists of extracting information from data sources to represent the elements of the recommendation system to generate their profiles. In our case, the elements are represented by FRs and NFRs. Before extracting the information through the Data Collector, we need to define the features capable of generating representative profiles of the elements in the problem domain. For this purpose, we elicited the knowledge of 3 scrum experts with experience between 4 and 10 years. They identified 5 features of software projects that may influence the definition of NFRs: project category (e.g., web, mobile, embedded, desktop, etc.), project domain (e.g., health, banking, etc.), project goal (e.g., product or prototype), project architecture (e.g., client-server, MVC, multilayered, etc.), and technologies (e.g., java, mongo, etc.).

For data collection, information about the FRs are retrieved from the SUSs and project profiles. More specifically, for each FR, information about FR category (high-level FR) and project profile are collected. For each NFR, NFR category and statements are extracted from the SUSs. Finally, information about the association between FRs and NFRs are collected.

2) Profile Manager

After collecting the data through the Data Collector, FR and NFR profiles are generated in the Profile Manager component. The profile of a FR $f \in F$ is composed by its FR category in addition to the 5 features of project p which it belongs to. In Table I, we present examples of FR profiles.

TABLE I: Examples of FR profiles.

ID	Proj. Categ.	Domain	Goal	Arch.	Tech. tags	FR categ.
f_1	Mob.	Home Auto.	Product	MVC	android, java, mongo	Login
f_2	Mob.	Home Auto.	Product	MVC	android, java, sqlite	Alarm and notif.
f_3	Mob.	Educat.	Product	MVC	android, jsoup, java, retrofit	Login
f_4	Web	Busin. info. sys.	Product	Client-server	nodejs,angular, webstorm	Login
f_5	Web	Busin. info. sys.	Prototype	multilayer	nodejs,angular, bootstrap	Status Vis.

NFR profiles present two features, i.e., NFR category and NFR statement. In Table II, we present examples of NFR profiles.

TABLE II: Examples of NFR profiles.

ID	NFR statement	NFR category
n_1	SSL encryption	Security
n_2	retrieved result must be paginated	Performance
n_3	access functionality with less than X clicks	Usability

Finally, FR profiles are complemented with the analysis of co-occurrences between FRs and NFRs based on information from the SUSs. The result of this analysis is a binary matrix. In Table III, we present an example of the matrix, in which the assigned value is 1, if a FR $f \in F$ has considered a NFR $n \in N$ or 0, otherwise.

TABLE III: Example of binary matrix that represents the co-occurrence between FRs and NFRs.

	n_1	n_2	n_3
f_1	1	0	1
f_2	0	0	1
f_3	1	0	1
f_4	1	0	0
f_5	0	1	1

After generating FR and NFR profiles, it is possible to generate customized recommendations of NFRs to FRs.

3) Recommender

The proposed recommendation system is based in the following characteristics:

- **memory-based collaborative filtering (neighborhood based technique) [9]**: recommendations are generated from the analysis of historical data on the co-occurrence between FRs and NFRs. To calculate the utility u of a NFR n for a FR f, we evaluate the relation of n with the k FRs (from previous projects) most similar to f;
- **recommendation of good items [9]**: the proposed recommendation system suggests a list with the j NFRs best suited to a given FR f.

To generate NFR recommendations, we carry out two activities: (i) estimate neighborhood using the k-Nearest Neighbors algorithm (kNN); and (ii) generate item recommendations.

For (i), we aim to identify the set of FRs $\hat{F}$ which includes the most similar FRs to a FR f_a, where f_a is the FR we intend to generate recommendations, called **target FR**. We use the kNN method to perform this task, since it returns the k nearest neighbors of an input element. Before applying the kNN, we perform a pre-filtering in the dataset to retrieve just those FRs of the same category of the target one. Then, the returned list of FRs is used as input of the kNN for the similarity calculation.

Before calculating similarities between FRs, we need to represent each FR profile through an *m*-dimensional feature vector, which enables the use of a similarity metric. We generate feature vectors based on the information extracted from the target FR profile, where each vector position refers to its features and is filled with a value (i.e., 0 or 1) according to the following condition: receives the value 1 if the feature is present in the FR profile, or 0, otherwise. In Table IV, we present an example of the vectors generated based on features extracted from a target FR f_a. In this example, it is possible to see that requirement f'_1 shares the same features of f_a, since its feature vector is filled only with values equal to 1. On the other hand, the functional requirement f'_3 differs from f_a in terms of "project domain" (i.e., Home Automation) and technology tag "mongo".

To calculate similarities among the feature vectors of f_a and pre-filtered FRs, we use the similarity based on Manhattan

TABLE IV: Examples of feature vectors generated based on features extracted from a target FR f_a.

	Mobile	Home Auto.	Product	MVC	*android*	*java*	*mongo*
f_a	1	1	1	1	1	1	1
f'_1	1	1	1	1	1	1	1
f'_3	1	0	1	1	1	1	0
f'_4	0	0	1	0	0	0	0

Distance (Equations 2 and 3), which is given by the sum of the differences between the values of the two input vectors of same dimension m. For example, the distance (Equation 2) between f_a and f'_3 is equal to 2, whereas the similarity (Equation 3) between them is 0.71 (see Table IV).

$$d(f_a, f') = \sum_{i=1}^{m} |f_{a_i} - f'_i|, \tag{2}$$

$$sim(f_a, f') = 1 - \frac{d(f_a, f')}{m}. \tag{3}$$

Therefore, to identify the k nearest neighbors of target FR f_a, we only have to sort in descending order the list of pre-filtered requirements by the calculated similarity, and return the first k items from that list. At the end of the process, we have the set with FRs $f' \in \hat{F}$ most similar to f_a.

The second activity is to generate item recommendations. More formally, we intend to recommend NFRs $n' \in N$ that maximize the utility of a target FR f_a. Thus, the value of the unknown utility $u_{f_a,n'}$ (Equation 4) for target FR f_a and NFR n' is computed as an aggregate of the amount of co-occurrence between the k neighbors of f_a and n', weighted by the similarity among f_a and its neighbors, where $u_{f',n'}$ returns 1 if NFR n' was considered in the development of neighbor FR f', or 0, otherwise.

$$u_{f_a,n'} = \sum_{f' \in \hat{F}} sim(f_a, f') \times u_{f',n'}. \tag{4}$$

Finally, we generate recommendations for target FR f_a as a list of NFRs $n' \in N$ sorted in descending order by the utility calculated for f_a and each NFR n', where $u_{f_a,n'} > 0$.

IV. Validation

To validate our approach, we executed an offline experiment following the instructions presented by Gunawardana and Shani [9], to answer the following research question: is it possible to automate the definition of NFRs in scrum-based projects based on historical data?

A. Experiment Goal

As mentioned before, the proposed recommendation system is based on collaborative filtering and item recommendation approaches. In this context, the most suitable metrics for evaluating our RS in an offline evaluation setup are related to precision and recall [9].

Therefore, the main goal of the experiment is to analyze the *proposed RS* for the purpose of *NFRs recommendation* with respect to the *precision and recall metrics* from the point of view of *a dataset generated with information provided by Scrum practitioners* in the context of *Scrum*.

B. Data Collection Procedure

To perform an offline experiment, it is necessary to use a dataset that corresponds as faithfully as possible to the real data of the problem domain. Thus, we conduct a survey with Scrum practitioners. To guarantee the reliability of the collected data, we only requested information that can be collected with the proposed instrumented Scrum.

The survey was responded by 12 Scrum project managers with 4 to 10 years of experience, who are mostly members of the same software company. At the end of the data collection, we gathered a dataset with the following attributes:

- 31 different software projects profiles, each one with its 5 features described;
- 31 different types of FR categories (e.g., "Login", "Status visualization", etc.);
- 47 different types of NFR statements (e.g., "SSL encryption", "retrieved result must be paginated", etc.);
- 130 functional requirements instances, where each instance represents a FR of a software project and a set of associated NFRs.

C. Offline Experimental Evaluation

In our experiment, we have two independent variables as input source and two dependent variable as output information. The first independent variable is represented by the number k of neighbors considered in the generation of recommendations, with 5 levels ($1, 3, 5, 7$ and 10). The second independent variable is represented by the number j of recommended NFRs that are considered to calculate the metrics, with 6 levels ($1, 3, 5, 7, 10$ and $|\hat{N}|$, where $\hat{N}$ represents the full list). Our two dependents variable are represented by the precision and the recall of the NFR recommendation achieved through a run based on a set of independent variables.

To perform the experiment we use the 10-fold cross-validation method, which randomly splits the dataset into 10 independent parts, and each part is used once as test set and the remaining as training set. Therefore, we separate the whole data into 90% for training and 10% for testing. Additionally, we repeat the execution for each set of independent variables. Thus, we have an amount of 300 runs.

D. Threats to Validity

The data collection was not done continuously during the Sprints as indicated in the Scrum instrumentation, which is a threat to internal validity. The size of the dataset is a threat to external validity, because it is required to have a large dataset to validate memory-based recommendation systems to represent different cases of the domain. We plan to address both threats in future work.

V. Results and Discussion

In Figure 4, we present the scatter plot of the two evaluated metrics (i.e., precision and recall), which summarizes the results obtained in the experiments. For example, the averages of precision and recall for the round with $k = 10$ and $j = 5$ are 44% and 73%, respectively.

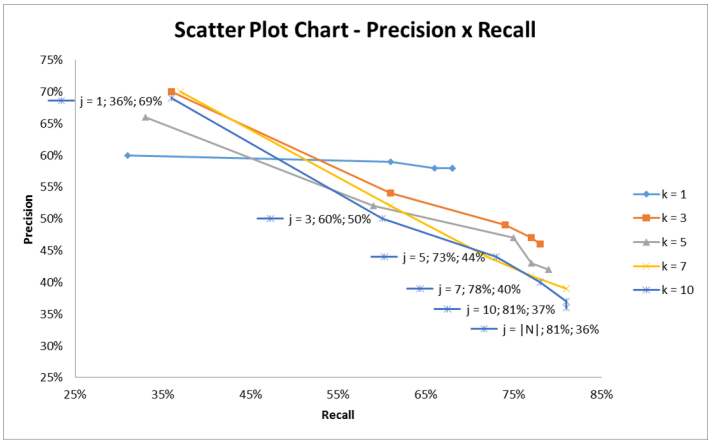

Fig. 4: Results obtained in the experiments.

Considering that the domain of NFRs recommendation is not sparse, since a FR is generally associated to a small number of NFRs, we conclude the results of the offline experiments are promising, since we observe rates of up to 81% of recall in the recommendations ($k = 7$ and $j = 10$), i.e., we correctly recommended 8 out of 10 NFRs raised in the dataset. Moreover, we observe recall rates greater than 70% and precision rates of up to 49% based on lists of recommendations with a size j equal to 5 ($k \in \{3, 5, 7, 10\}$), which can be easily handled by Scrum Teams at Sprint Planning meetings and support the early definition of NFRs. We also notice that the ideal number of neighbors k is greater than or equal to 3.

Finally, we conclude that it is possible to automate the definition of NFRs in scrum-based projects based on historical data. Furthermore, the precision observed in experiments can be improved with a larger dataset, since the proposed recommendation system is memory-based and the quality of recommendations depends on the representativeness of the dataset.

VI. Conclusion and Future Work

In this paper, we proposed a NFRs recommendation system to support Scrum practitioners to consider NFRs early in the process. The proposed solution is divided into two main steps, a Scrum instrumentation and a recommendation system. The instrumentation contributes to the data collection process in Scrum-based projects and can be used for different application domains. The NFRs recommender uses historical data from Scrum-based projects, which was not seen in previous works in this research field. Therefore, our work can be used as baseline for future works that intend to investigate this subject.

The offline experimental evaluation showed the feasibility of automating the definition of NFRs through historical data of Scrum projects. We observed an average recall rate of up to 81%, which is promising. Although we observed values of less than desired precision, we believe that the results can be improved with a larger dataset.

For future work, we intend to keep the data collection process to increase the dataset and improve its representativeness, and then, replicate experiments. Additionally, we intend to carry out a case study with running Scrum-based projects to evaluate recommendations in online environments.

Acknowledgment

The authors would like to thank CAPES for supporting this work.

References

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.

[2] V. Bajpai and R. P. Gorthi. On non-functional requirements: A survey. In *2012 IEEE Students' Conference on EECS*, pages 1–4, March 2012.

[3] M. Bourimi, T. Barth, J. M. Haake, B. Ueberschär, and D. Kesdogan. AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-technical systems following agile methodologies. *Lecture Notes in Computer Science*, 6409 LNCS:182–189, 2010.

[4] T. Dingsyr, S. Nerur, V. Balijepally, and N. B. Moe. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6):1213 – 1221, 2012. Special Issue: Agile Development.

[5] D. Domah and F. J. Mitropoulos. The nerv methodology: A lightweight process for addressing non-functional requirements in agile software development. In *SoutheastCon 2015*, pages 1–7, April 2015.

[6] W. M. Farid. The normap methodology: Lightweight engineering of non-functional requirements for agile processes. In *Proceedings of the 2012 19th APSEC - Volume 01*, APSEC '12, pages 322–325, Washington, DC, USA, 2012. IEEE Computer Society.

[7] W. M. Farid and F. J. Mitropoulos. Normatic: A visual tool for modeling non-functional requirements in agile processes. In *2012 Proceedings of IEEE Southeastcon*, pages 1–8, March 2012.

[8] A. Firdaus, I. Ghani, D. N. Abg Jawawi, and W. M. N. Wan Kadir. Non functional requirements (NFRs) traceability metamodel for agile development. *Jurnal Teknologi*, 77(9):115–125, 2015.

[9] A. Gunawardana and G. Shani. A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *The Journal of Machine Learning Research*, 10:2935–2962, 2009.

[10] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, predicting and prioritizing (cepp) non-functional requirements metadata during the early stages of agile software development. In *SoutheastCon 2015*, pages 1–8, April 2015.

[11] R. R. Maiti and F. J. Mitropoulos. Capturing, eliciting, and prioritizing (cep) nfrs in agile software engineering. In *SoutheastCon 2017*, pages 1–7, March 2017.

[12] R. R. Maiti and F. J. Mitropoulos. Prioritizing Non-Functional Requirements in Agile Software Engineering. *Proceedings of the SouthEast Conference on - ACM SE '17*, pages 212–214, 2017.

[13] A. E. Sabry and S. S. El-Rabbat. Proposed framework for handling architectural nfr's within scrum methodology. In *Proceedings of the International Conference on SERP*, page 238. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2015.

[14] V. Sachdeva and L. Chung. Handling non-functional requirements for big data and iot projects in scrum. In *2017 7th International Conference on Cloud Computing, Data Science Engineering - Confluence*, pages 216–221, Jan 2017.

[15] T. Suryawanshi and G. Rao. A Survey to Support NFRs in Agile Software Development Process. 6(6):5487–5489, 2015.