A Search-based Software Engineering Approach to Support Multiple Team Formation for Scrum Projects

Alexandre Costa
Federal University of Campina Grande
Intelligent Software Engineering Group
antonioalexandre@copin.ufcg.edu.br

Felipe Ramos
Federal University of Campina Grande
Intelligent Software Engineering Group
feliperamos@copin.ufcg.edu.br

Mirko Perkusich
Federal Institute of Paraiba
mirko.perkusich@ifpb.edu.br

Arthur Freire
Federal University of Campina Grande
Intelligent Software Engineering Group
arthurfreire@copin.ufcg.edu.br

Hyggo Almeida
Federal University of Campina Grande
Intelligent Software Engineering Group
hyggo@dsc.ufcg.edu.br

Angelo Perkusich
Federal University of Campina Grande
Intelligent Software Engineering Group
perkusic@dee.ufcg.edu.br

Abstract—**Search-Based software engineering (SBSE) deals with metaheuristic search-based optimization techniques to provide solutions for complex problems. A popular problem in literature is the team formation problem (TFP), which consists of finding the best allocation of human resources to a software development project. This problem is recognized as NP-hard and it is more complex in companies that carry out multiple projects. This paper presents an effective and automated approach to allocate multiple developers into multiple teams to maximize the technical compatibility between them. The approach consists of an SBSE method that uses Genetic Algorithm to simultaneously build multiple teams, using data from tag-based profiles. We conducted an empirical evaluation using data from eight real-world software projects of a Brazilian company. The results indicate that tag-based profiles is a promising information source to represent technical knowledge, since the suggested teams were considered to have the proper skills to the attend the technical demand of the projects. The approach was able to reach high levels of satisfaction, delivering teams in an effective and automated way. Although, further investigation needs to be conducted to reach stronger conclusions.**

I. Introduction

Search-Based Software Engineering (SBSE) is a field that deals with metaheuristic search-based optimization techniques to provide automated and semi-automated solutions for complex problems in Software Engineering (SE) [6], [5], [7]. According to Harman et. al. [7], typical SE problems that involve testing, design, requirements engineering, management, and refactoring can be well adapted for SBSE and have been successfully formulated as search-based optimization problems. For instance, the team formation problem (TFP), which consists of finding the best allocation of resources (developers) to a software development project. This problem has gain special attention over the last years and became a well-researched topic in the literature [1], [4], [11], [2].

The TFP is recognized as NP-hard [13] and it is more complex in companies that carry out multiple projects. It is necessary to optimize the allocation of multiple developers with different sets of skills to multiple projects with different sets of requirements. The goal is to optimize each projects' chances of success considering the limited resources available and attempt to have all projects succeeding. The previously described scenario represents a multiple team formation problem (MTFP) [4].

Finding optimal teams brings special concern with agile methods, such as Scrum [10], a team-centered framework designed to deliver products with the highest possible value. In Scrum, the developer team is a self-organized and multidisciplinary group which means that, among others, the team must have the all the required technical knowledge to deliver the product. A nine-month field study of professional developers in a Scrum team [8] found that highly specialized skills and a corresponding division of work are the most important barrier for achieving effective teamwork. Therefore, it is necessary to make the most of the available resources, i.e., choosing right teams to the right projects.

In this paper, we present a SBSE approach to form multiple teams for Scrum projects. The approach is divided into two parts. First, we learn the technical knowledge of the developers through information extracted from the projects in which they have worked. For this purpose, we instrument the Scrum process by creating tag-based profiles for developers and projects. Therefore, we complement the Scrum framework with activities to assign tags to artifacts, i.e., the Product Backlog (PB), User Stories (US) and Technical Tasks. Second, we build a data structure to run a Genetic Algorithm to allocate developers to projects, forming teams with maximum technical compatibility, considering the projects demands and the developer's skills. We chose Genetic Algorithm, because it one of the most suitable methods for combinatorial optimization problems [3].

To evaluate our solution, we used real-world data from eight Scrum software projects and we collected data from

DOI reference number: 10.18293/SEKE2018-108

16 developers and 4 projects managers. The results indicates that the tag-based profiles are a useful information source to support the multiple team formation with a high level of satisfaction.

This paper is organized as follows. Section II presents related work on SBSE applied to MTFP. Section III presents our solution to mitigate the multiple team formation problem. Section IV presents the design and discusses the results of the empirical evaluation. Section V presents the validity threats; and Section VI presents the final remarks, limitations and future work.

II. Related Work

As mentioned before, a particular case in the TFP is the MTFP, which consists of simultaneously allocating multiple developers to multiple projects to maximize attributes from both developers and projects.

Palacios et. Al. [1] presented a recommender system designed to assist project managers in configuring multiple teams for work packages defined by Scrum Projects. The system is based on fuzzy logic, rough set theory and semantic technologies. In this work each project manager is responsible for building both project and developers profiles using competence baseline documents.

Strnad and Guid [11] proposed a decision support system based on Fuzzy Logic and Genetic Algorithm. The system uses fuzzification to automatically obtain fuzzy skill assessments from numerical data of an employee database. Genetic Algorithms are used to optimize the teams formations according to the projects requirements, which are obtained by a standard document specification inside company.

Silva and Costa [2] presented a framework based on dynamic programming to allocate human resources in multiple information system projects. The main goal is to determine the fit between the complete set of skills available from a candidate member of a project team and the skills required for that project to minimize the time required to complete a project demand.

Gutierrez [4] proposes an optimization model using a quadratic objective function, linear constraints and integer variables. The optimization model is solved by three algorithms: a Constraint Programming approach provided by a commercial solver, a Local Search heuristic and a Variable Neighborhood Search metaheuristic. Sociometric techniques are used to estimate performance characteristics such as productivity, training ability, leadership, efficiency, among others.

Ren el. al. [9] proposed a search-based software project method to build teams based on Cooperative Co-evolution. The teams are formed aiming to reduce the required time to complete the projects work packages.

The main difference between the cited works and ours, is the way the profiles of developers and projects are built. Most of the approaches depend on the knowledge and feeling of the project managers to build the profiles. For this reason, the results are subjective. In our approach, the profiles are derived from the technologies assigned to the Scrum artifacts during the Scrum Instrumentation, minimizing the subjectivity. Another difference is that the profile information grows dynamically, because the instrumentation is refined throughout the project.

III. Proposed Approach

The proposed solution is divided into two parts: Scrum Instrumentation and Solution Operationalization, which are detailed as follows.

A. Scrum Instrumentation

Scrum is a framework for developing, delivering, and sustaining complex products [12]. It was designed to be complemented with technical and managerial processes as needed. We complement Scrum by applying tags to the artifacts to register the technical skills necessary to develop the features associated with them. These skills corresponds to programming languages, frameworks, platforms, APIs, architectures, databases, or any technology or technical knowledge necessary to develop a feature.

To lead the tag labeling process, a new role is proposed: (Scrum) Tagger. The Tagger is responsible for all the tag labeling process. To assure a standardized tags assignment, it is recommended that the company establish a small team of Scrum Taggers who should define specific rules to avoid that the same technical skill is reperesented using different tags in different projects. The Scrum Instrumentation occurs throughout the project and includes the following labeling processes:

1) Product Backlog Labeling: the Tagger starts to work during the initial PB definition (Figure 1[1]). The PB is an ordered list of the product requirements. The Product Owner (PO) is the responsible for building the PB. Therefore, the Tagger conducts informal conversations with the PO to elicit all or at least most of the technologies necessary to develop the PB. Then, these technologies are converted into tags, forming the Backlog Tag set.

2) User Story Labeling: the labeling process occurs during Sprint Planning meetings (Figure 1[2]), which is an event where some requirements are selected from the PB to be developed during the given Sprint, defining the Sprint Backlog. This requirements are usually represented by User Stories. The Tagger, together with the Development Team, labels all the USs of the Sprint Backlog. For this purpose, the Tagger consults the team regarding the necessary technical skills to develop the USs. Then, the Tagger converts the skills into tags, forming the User Story Tag Set. It is important to note that this set is a subset of the Backlog Tag, therefore, if a new tag appears during this step the Backlog Tag set is updated.

3) Task Labeling: to define the Sprint Backlog, the USs are splited into Technical Tasks. When a task is done, the responsible selects a subset of tags from the User Story Tag set. The selected subset, called Task Tag set, represents the technical skills demanded to develop the feature represented by the task. This labeling process (Figure 1[3]) occurs throughout the Sprints. If necessary, the developer can suggest new tags, which can be reviewed by the Tagger after the end of Sprint (Figure 1[4]). If new tags are created, the User Story Tag and the Backlog Tag sets are updated.

B. Solution Operationalization

The second part of the proposed approach consist of creating the data structure necessary to run a Genetic Algorithm to allocate multiple developers into multiple projects. First, tag-based profiles are created for both developers and projects from the data acquired from the Scrum Instrumentation. Then, technical compatibilities between developers and projects are calculated using feature vectors. Lastly, we use weighted similarity coefficients to create a technical compatibility metric that is used in the Genetic Algorithm's fitness function to find the optimal team formation for each project. These steps are detailed as follows.

1) Tag-based profile building: The developer profile is defined from two sources of information: Curriculum and Development History. First, technical knowledge documented in the developers' curriculum is converted into weighted tags. The weight corresponds to the period of experience with the given technology. For example, if the developer worked with the Java programming language for 18 months, his profile will contain a tag *java(18)*. Second, during the Sprint, the developers works in tasks in which tags were applied during the Scrum Instrumentation. Over time, the developer profile is being formed by the tags that were applied to the tasks completed by him. These tags also have weight, in this case it is given by the number of times a tag appeared in a task. For example, if during the Sprint the developer completed five tasks with a java tag applied, his profile is filled with the weighted tag *java(5)*.

The project profile is built from the Product Backlog Tag set. As the developer profile, it grows throught the execution of the project, as the project requirements change and the product backlog items are more detailed.

2) Feature vector: A feature vector is a k dimensional vector composed by numerical values and represents a set of the object's characteristics. The feature vectors are created from tag-based profiles. They are necessary to calculate the similarity between profiles.

The project feature vector is called Backlog Vector (V_B). Each tag presented in the project profile corresponds to an index in the vector. Note that different projects may have different profiles, consequently, they have different feature vectors.

For each developer, two feature vectors are created from his profile. The Curriculum Vector (V_C) is built from the Curriculum Source; the Development Vector (V_D), from the Development History Source. Both vectors use the same structure as the project feature vector to allow the similarity calculation.

3) Developer Feature Vector Normalization: Since the weight of the tags that compose the developer feature vectors come from two different data sources with different magnitudes, it is necessary to harmonize the scales by normalizing it to values between 0 (zero) and 1 (one). The normalization uses the Equation 1, where $x = (x_1, x_2, x_3, ..., x_k)$ is the key value in the same position at each developer feature vector and z_i is the i_{th} normalized value. Note that there are two normalization processes, one for each developer feature vector.

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{1}$$

4) Similarity calculation: We used Manhattan Similarity to calculate the similarity between two k-dimensional feature vectors. The similarities are used to determine the technical compatibility between the developer and project profiles. Between each profile, two different similarity coefficients are calculated. One represents the similarity between the Backlog Vector and Curriculum Vector and the other represents the similarity between the Backlog Vector and Development Vector. Before the similarity calculation, the Backlog Vector is filled with 1s (ones) to represent a maximum technical demand of a project, i.e., the developer feature vectors, which are composed by values closer to 1s (ones), will have higher values of similarity.

$$1 - \frac{\sum_{i=1}^{k} |u[i] - v[i]|}{k} \tag{2}$$

5) Genetic Algorithm to Form Multiple Teams: The multiple team formation process consists of allocating multiple developers into multiple project team to maximize the technical compatibility between the project demands and the developer skills. For this purpose, we propose a metric called Technical Compatibility Level (TCL). The TCL is generated based on a weighted similarity coefficient given by the Equation 3. The α represents the weight of the similarity calculated between the Backlog and Curriculum vectors [$sim(v_B, v_C)$]. The β represents the weight of the similarity calculated between the Backlog and Development vectors [$sim(v_B, v_D)$]. For each project and developer, a TCL is generated.

$$TCL = \frac{\alpha \cdot sim(v_B, v_C) + \beta \cdot sim(v_B, v_D)}{2} \tag{3}$$

An example of the chromosome structure used in the Genetic Algorithm is presented in Figure 2. In this case, we have 3 projects and 12 developers. The genes represent the developers and, depending on his position in the structure, a different TCL is used during the fitness calculation. For instance, if a developer D1 appears in the indexes between 0 and 4, the TCL between D1 and the Project A is used. If D1 appears in the indexes between 5 and 7, the TCL between D1 and the Project B is used. The goal of the fitness function is to find a candidate solution, where the sum of the TCLs corresponded to each gene is maximized. Consequently, an optimal solution corresponds to a configuration where the matching represents maximum technical compatibility.

IV. Empirical Evaluation

To evaluate our approach, we conducted an empirical evaluation in a Brazilian software development company, in

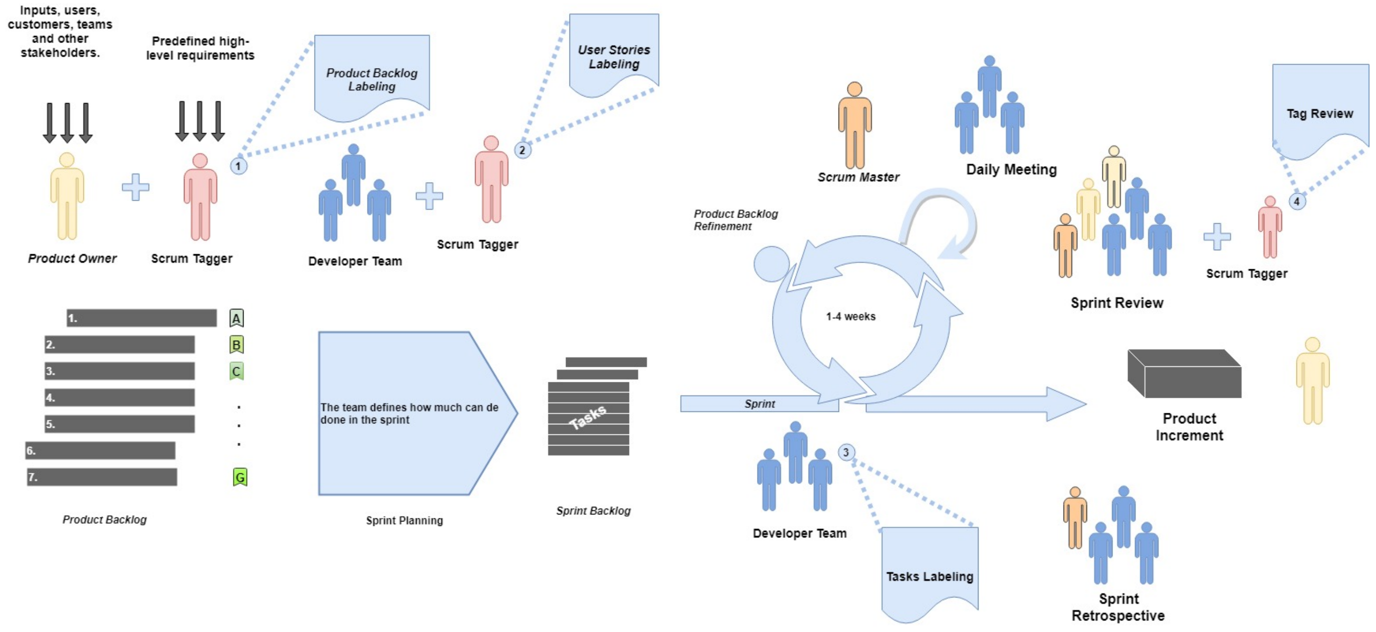

Fig. 1. Scrum Instrumentation

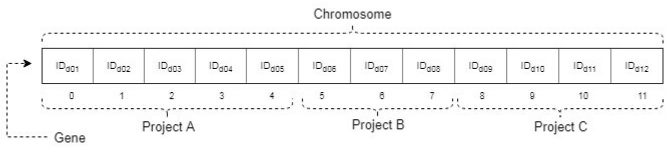

Fig. 2. Example of a chromosome used in the Genetic Algorithm

which we collected real-world data from eight Scrum software projects. Four of them (P1, P2, P3 and P4) were running during the evaluation and the remaining projects (P5, P6, P7 and P8) were already finished. Each of the running projects is composed by a project manager (PM) and four junior developers. The PMs know the profiles of all 16 developers, since there is a regular rotation among the projects, to avoid knowledge islands. The PMs have 2-5 years of experience in Scrum projects and the developers 1-3 years in software development.

In Table I, we show statistics regarding the projects. The first column contains the identification of the running projects; the second contains the number of Sprints executed in the project. Each Sprint lasted 2 weeks; the third presents the number of User Stories completed; the fourth is the number of tasks done (each task was performed individually); the fifth shows the identification of the team members; the sixth, contains the Backlog Tag set collected during the instrumentation process performed in the evaluation. For reasons of confidentiality, the projects and developers identifications were presented in an anonymous way.

We performed the instrumentation process with the four running projects (P1, P2, P3 and P4). The instrumentation was designed to be executed in a interactive and incremental way during the Scrum Events and throughout the project. Unfortunately, the projects were already started and we had to apply the process at once to retrieve the previous data of the projects. For each project, we applied the three labeling processes described in Section III-A. To avoid overloading of the participants, which could compromise the process, we divided the labeling in turns. During the morning, the team performed the Backlog and User Story labeling processes and during the afternoon the Tasks' one. The instrumentation was applied in different days for each project. The Scrum Tagger role was performed by the first author of this paper. In Table I, we can see the Backlog Tag set of projects P1, P2, P3 and P4.

To create the Curriculum Source, we provided an online questionnaire and all developers were guided to respond to it. The questionnaire was composed by the tags present in the Backlog Tag set (Table I) and complemented with tags that represent popular technologies which not appeared during the instrumentation. Also, the developers were free to add other technologies he have worked outside the company by answering an open-ended question. Naturally, duplicated tags were excluded from the questionnaire. For each tag, the developers were asked to register their experience (in months) with the given technology.

To evaluate the approach, we simulated a scenario in which four projects (P1, P2, P3 and P4) were used to learn the profile of the developers with the goal of allocating them to other four projects (P5, P6, P7 and P8). We present the Backlog Tag set for projects P5, P6, P7 and P8 in Table II. To collect the data for projects P5, P6, P7 and P8, we only executed the Product Backlog labeling process.

We executed our approach varying α and β weights and the results are presented in Table III. For each new project, we

TABLE I
DATASET STATISTICS

Project	*#Sprints*	*#User Stories*	#Tasks	Team Formation	Backlog Tag set
P1	6	41	136	D3, D7, D9, D14	angular-charts, angular-material, angularjs, bower, Chart.js, checkstyle, css, ESlint, express, Firebase, Gitlab, gulp, html, javascript, jenkins, JSHint, json, Mocha, MongoDB, node.js, pmd
P2	4	15	97	D1, D2, D10, D11	angular-charts, angular-material, angularjs, bower, Chart.js, checkstyle, css, cytoscape, eslint, findBugs, gitlab, gson, hamcrest, html, java, javascript, jpa, JSHint, json, junit, maven, mockito, mySQL, PMD, spring-boot, spring-jpa, swagger
P3	3	12	63	D4, D5, D6, D8	angular-charts, angularjs, bootstrap, bower, chart.js, css, eclipse, gitlab, gulp, html, http-request, java, javascript, javascript, javax, json, maven, mysql, npm, spring-boot, spring-jpa, sts, swagger, webstorm
P4	7	21	76	D12, D13, D15, D16	android, angular-material, angularjs, http-request, Beaglebone, Bonescript, Bootstrap, C++, Chart.Js, Css, Express, Gitlab, Heroku, Html, iot, jade, java, javascript, material-design, MongoDB, node.js, npm, parse-server, python, Raspberry, Socket.io, XML

TABLE II
PROJECT'S BACKLOG TAG SET

Project	Backlog Tag set
P5	android, android-studio, Beaglebone, firebase, git, glide, http-request, iot, java, javascript, node.js, parse-server, postman, sqlite, xml, zxing
P6	android, java, xml, node.js, volley, firebase, butterknife, Body-parser, Json, Express, Mongoose, Nodemailer, Validator, webstorm, javascript
P7	Body-parser, node.js, bootstrap, css, express, Gitlab, html, http-request, javascript, JSHint, json, jwt, MongoDB, Mongoose, Nodemailer, npm, webstorm, git
P8	angularjs, Body-Parser, bootstrap, bower, css, cytoscape, ESlint, express, git, Gitlab, gulp, html, http-request, jasmine, javascript, jenkins, JQuery, JSHint, json, karma, material-design, Mocha, MongoDB, Mongoose, bootstrap, node.js, NPM, tslint, typescript,Postman

formed five teams and presented to the corresponding project managers. The PM1 was manager of P8, PM2 was manager of P7, PM3 was manager P5, PM4 was manager P6. Then, we asked them to answer a couple of questions. First, the PM was required to rank the five teams suggested, according to the level of suitability to the project, i.e, the better ranked teams should be those who posses the set of technical skill that best meets the demands of the project. Second, they should rate using five point Likert scale their satisfaction with the best ranked team(s) (i.e., multiple teams could be considered tied with the best rank).

In Figure 3, we show the results related to the first question. It indicates that when the weight of the Curriculum source of the developer is high the team is ranked in the best position by most of the managers. When we increase the weight of the Development History source the suggested teams start to be ranked in the last positions. Althought the Curriculum may not be the most reliable information source, it has the bigger amount of data, since it represents the knowledge accomplished by the developers during all their professional journey. On the other hand, the volume of the Development History source is much smaller, because the data collected came only from the projects the developers were participating during the evaluation and it corresponded just to a few months of development. This indicates that Curriculum data reflected most of the technical knowledge the developers claimed to have, at least by the PMs expectations. Unfortunately, we can not make solid conclusions about the Development History source, because of its small amount of data. We believe that, at a given point after collecting data, this source is more trustworthy than the Curriculum, but need to collect more data to verify this hypothesis. In Figure 4, we present the results related to the second question, which show that the PMs were satisfied with the allocations.

V. THREATS TO VALIDITY

We identified a few threats to validation in our work. As same as others agile methodologies, Scrum stands for individuals and interactions over processes and tools. We proposed a Scrum instrumentation as an incremental and interactive process. It is designed to be less intrusive as possible to Scrum framework, since it occurs during already existents events and demands just a few more steps. Althought the process was carefully designed, we could not apply it as it is suppose to be, because the projects had already started. So, we applied the process at once in each project. We consider this to be an internal threat to validity.

Since we only collected data from one company for a short period of time, we cannot claim external validity. We aim to address this threat in future work.

VI. CONCLUSIONS

In this paper, we propose a SBSE approach to support multiple team formation for Scrum projects. Among our contributions, we can highlight the Scrum Instrumentation process, which allows the creation of tag-based profiles for developers and projects. These profiles can be used to assist technical knowledge management. Since, the instrumentation process is designed to be incremental and interactive, the profiles are susceptible to reflect changes during the project execution and grow gradually. We can also emphasize the creation of an automated method based on Genetic Algorithm to support the project managers during the simultaneously allocation of multiple developers into multiple software projects, forming teams with maximum technical compatibility.

TABLE III
OBTAINED RESULTS FROM THE VARIATION OF α AND β WEIGHTS

	P5	**P6**	**P7**	**P8**
$\alpha = 100\%$ and $\beta = 0\%$	D4, D12, D13, D15	D7, D8, D9, D14	D1, D2, D3, D6	D5, D10, D11, D16
$\alpha = 75\%$ and $\beta = 25\%$	D7, D12, D13, D15	D4, D8, D9, D14	D1, D2, D3, D6	D5, D10, D11, D16
$\alpha = 50\%$ and $\beta = 50\%$	D7, D12, D13, D15	D4, D8, D9, D16	D1, D2, D3, D6	D5, D10, D11, D14
$\alpha = 25\%$ and $\beta = 75\%$	D7, D12, D13, D15	D2, D4, D8, D16	D3, D5, D6, D14	D1, D9, D10, D11
$\alpha = 0\%$ and $\beta = 100\%$	D7, D12, D13, D16	D2, D4, D8, D11	D3, D5, D14, D15	D1, D6, D9, D10

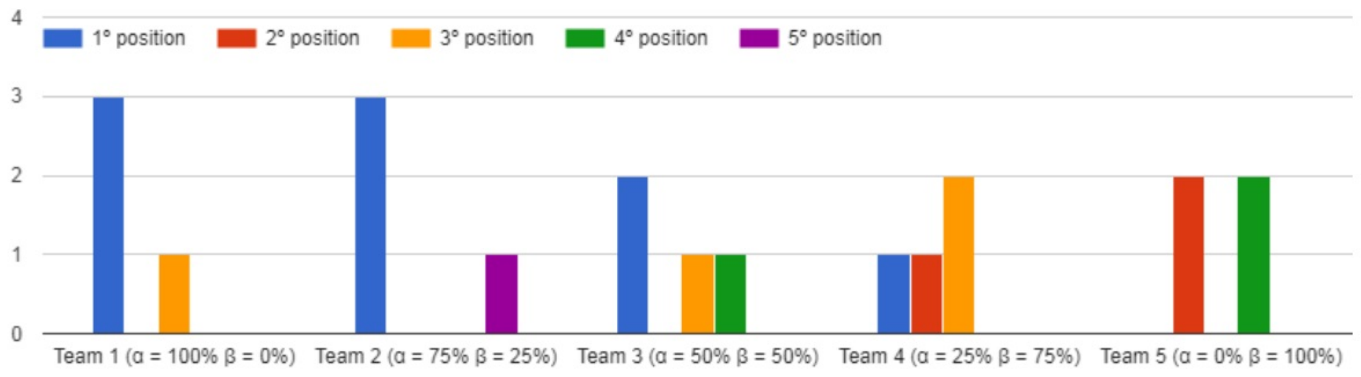

Fig. 3. Ranked teams by the projects managers

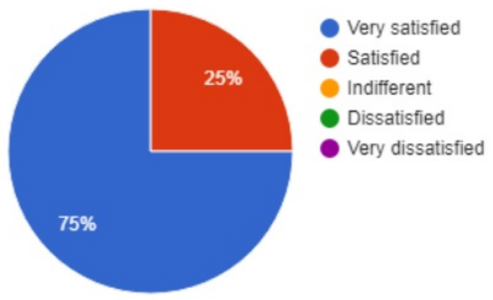

Fig. 4. Project managers satisfaction to the better ranked teams

As a limitation of our approach, we can point the slow growth of the Development History data source. Since this source depends on the developer participation in software projects of the company, the amount and diversity of these data might need time to reach the same size as the Curriculum source. We hypothesize that the Development History is a more reliable source, but we could not verify it in this study. We plan to verify it in future work, by increasing the number of projects and developers for the next empirical evaluation and test different scenarios. Also, we intend to determine optimum values for α and β dynamically, according to the information volume of each data source. Furthermore, we plan to integrate our tagging mechanism to quality and productivity indicators, to have a more reliable regarding the knowledge (i.e., expected performance) of the developers.

REFERENCES

[1] R. Colomo-Palacios, I. González-Carrasco, J. L. López-Cuadrado, and Á. García-Crespo. Resyster: A hybrid recommender system for scrum team roles based on fuzzy and rough sets. *International Journal of Applied Mathematics and Computer Science*, 22(4):801–816, 2012.
[2] L. C. e Silva and A. P. C. S. Costa. Decision model for allocating human resources in information system projects. *International Journal of Project Management*, 31(1):100–108, 2013.
[3] E. Falkenauer. *Genetic algorithms and grouping problems*. Wiley New York, 1998.
[4] J. H. Gutiérrez, C. A. Astudillo, P. Ballesteros-Pérez, D. Mora-Melià, and A. Candia-Véjar. The multiple team formation problem using sociometry. *Computers & Operations Research*, 75:150–162, 2016.
[5] M. Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, pages 342–357. IEEE Computer Society, 2007.
[6] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833 – 839, 2001.
[7] M. Harman, S. A. Mansouri, and Y. Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.*, 45(1):11:1–11:61, Dec. 2012.
[8] N. B. Moe, T. Dingsøyr, and T. Dybå. A teamwork model for understanding an agile team: A case study of a scrum project. *Information and Software Technology*, 52(5):480–491, 2010.
[9] J. Ren, M. Harman, and M. Di Penta. Cooperative co-evolutionary optimization of software project staff assignments and job scheduling. *Search Based Software Engineering*, pages 127–141, 2011.
[10] K. Schwaber and M. Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.
[11] D. Strnad and N. Guid. A fuzzy-genetic decision support system for project team formation. *Applied soft computing*, 10(4):1178–1187, 2010.
[12] J. Sutherland and K. Schwaber. The scrum guide. *The definitive guide to scrum: The rules of the game. Scrum. org*, 268, 2013.
[13] G. J. Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial OptimizationEureka, You Shrink!*, pages 185–207. Springer, 2003.