Automated Software Engineering (2022) 29:51
https://doi.org/10.1007/s10515-022-00357-7

Quantifying effectiveness of team recommendation for collaborative software development

Noppadol Assavakamhaenghan[1] · Waralee Tanaphantaruk[1] · Ponlakit Suwanworaboon[1] · Morakot Choetkiertikul[1] · Suppawong Tuarob[1]

Received: 27 August 2021 / Accepted: 8 August 2022 / Published online: 20 August 2022

Abstract

It is undeniable that software development is a team-based activity. The quality of the delivered product highly depends on the team configuration. However, selecting an appropriate team to complete a software task is non-trivial, as it needs to consider team compatibility in multiple aspects. While extensive literature introduced multiple team recommendation algorithms, such algorithms are not designed to support the specific roles in software teams. This paper proposes a novel set of metrics for measuring five dimensions of a software team's effectiveness, including historical collaboration, team cohesiveness, teammate interaction, team members' expertise, and role experience. Furthermore, Wining Experience-based Software Team RECommendation (WESTREC) is introduced to solve the software team recommendation problem. WESTREC considers multiple aspects of team characteristics, including historical collaboration, team cohesiveness, teammate interaction, project description, team members' expertise, and role experience. Specifically, given a software project, a machine learning based team scoring function is used along with the Max-Logit algorithm to approximate and recommend suitable software team configurations for the given task. We validate the effectiveness of the WESTREC on real-world software development datasets (i.e., Atlassian and Apache). Furthermore, we study the factors that affect the performance of collaborative software development and propose a method to evaluate the effectiveness of a software team. The results show that WESTREC outperforms state-of-the-art baseline approaches in three out of five groups of team effectiveness metrics associated with different team characteristics in large software systems. Our research findings not only illustrate the efficacy of automatic software team evaluation using machine learning techniques but also serve as building blocks for potential applications that involve automatic team formation and evaluation, such as automatic recommendation of research collaborators and grouping personnel for team-based projects.

Extended author information available on the last page of the article

Keywords Collaborative software development · Machine learning · Knowledge graph · Software team effectiveness · Team recommendation

1 Introduction

Software development can be considered the creation of socio-technical systems consisting of both human and technological aspects. Meanwhile, the problems and required work associated with software development are various and complex, requiring multiple sets of technical skills and roles to complete a job (Dingsøyr and Dybå 2012; Dubinsky and Hazzan 2004; Zhu et al. Nov 2006; Yasrab et al. 2011). Thus, a collaboration of many software practitioners with appropriate technical skills and compatibility is crucial to developing successful software (Wick 1999; Chow et al. 2008; Han and Huang 2007). According to CHAOS reports from Standish Group, the competence of team members is one of the key factors in determining the success of the software project because effective teamwork and skillful team members have shown to contribute to the ability to handle challenging circumstances (The Standish Group 2015).

However, manually arranging team members is a challenging task that is typically done by experienced project leaders since many aspects, such as technical skills, roles, and team collaboration, must be considered. The survey from Standish Group's CHAOS research shows that 27% of the process of selecting and approving participants in a software project is done manually by a project leader or a senior member. 50% of project selection is based on an ad-hoc process that is specifically crafted for each organization. A highly experienced project manager believed creating a new team was probably the most important, yet the most difficult, because it involved a high level of detail in setting up the team (Oliver Bossert and Kretzberg 2018). However, such a manual team member selection process may not efficiently apply to large-scale, modern, especially open-source software projects due to the following reasons:

- **Dynamic Users**: In large-scale, open-source software systems such as Apache[1] and Atlassian[2], the pool of available software practitioners and their skills are dynamically changing. It would be a tedious task for a project leader to constantly keep abreast of available candidate members and their current skill sets while maintaining multiple fast-paced micro software projects.
- **Lack of Human Resource Experience**: In contrast with formal projects where project leaders are trained to equip themselves with human resource management skills (Foster 2014), many projects in an open-source software system are small, aiming to incrementally improve existing projects by implementing more features, fixing bugs, etc., and led by experienced software developers who may not have extensive skills in human resource management. While these project leaders

[1] https://www.apache.org/.
[2] https://www.atlassian.com/.

may be able to validate each member's technical experience, their ability to vet a candidate member that is compatible with the rest of the team could fall short.

While software tracking tools have emerged to assist the collaborative development of software activities, issues still exist, such as delays and interruptions that lead to project reopening, which could indicate the ineffectiveness of the team member configuration. This situation can be caused by many factors such as lack of essential experience or skills and team communication problems (Assavakamhaenghan et al. 2019; Tuarob et al. 2021). Such problems call for a decision support system capable of selecting team members, considering their relevant experience, skills, and teamwork compatibility.

Existing algorithms have been developed that automatically recommend software practitioners for certain functions, including developers, assignees, and reviewers (Naguib et al. 2013; Tian et al. 2016). Nevertheless, contemporary software issues are resolved by a number of techniques, including developing, reviewing, and testing. Consequently, an effective software team must include personnel responsible for these various functions. Incompatible team arrangements (e.g., teams that have never worked together in the past) may also lead to a high-impact risk (Jiang et al. 2000). Research conducted by Fagerholm et al. (2014) and Günsel et al. (2012) reveals that team-related factors, including team communication capability, flexibility, and member compatibility, are essential to the success of software projects. However, the present approaches for suggesting certain roles cannot take these considerations into account.

Indeed, team formation algorithms were independently proposed, but not so many of them are directly related to the software team (Wang et al. 2015). One possible solution to this problem is to use a machine learning approach to combine individual strengths and team features to recommend a team (Liu et al. 2014). However, there are limitations to this approach because this method uses only a small set of features to characterize teams that are not sufficiently designed for collaborative software development tasks. Another solution is T-RecS, a team recommendation system that focuses on expertise and cohesiveness (Datta et al. 2011b). This system makes recommendations for research teams by considering researchers' skills for each area as well as social interactions. Through this system, a candidate team is provided with a score to determine if the team is suitable for the given task or not. However, their method does not have a team formulation algorithm, so it needs to generate all possible combinations of team configurations. Although this technique is designed to solve the team recommendation problem, it only focuses on the research collaboration applications and, therefore, is not directly applicable to collaborative software development applications.

Therefore, to address the software team recommendation problem, we propose a novel set of features representing multiple aspects of an effective software team, such as relevant experience, technical skills, and team compatibility. These features are used to compose a scheme for software team effectiveness measurement, *STEM*, that reflects five dimensions of software team effectiveness, including communication cost, team morale, team cohesion, team expertise, and team proficiency. Furthermore, to illustrate the application of the features, WESTREC (Wining

Experience-based Software Team RECommendation) is proposed, which incorporates the proposed features to an existing Max-Logit-based team recommendation algorithm (Liu et al. 2014). WESTREC incorporates many machine learning techniques in many different parts of its process (i.e., topic modeling in finding vectors to represent a project description, sentiment analysis in the comments in a project, and binary classification machine learning model for team scoring function). Then, WESTREC is empirically evaluated on the two well-known open-source software datasets (i.e., Apache and Atlassian). The results show that WESTREC outperforms the other baseline approach (i.e., Recentness and Liu et al. (2014)'s approach) in three out of five groups of metrics associated with different characteristics of teams in a larger software system. Concretely, our research contributions are as follows:

- We propose a novel set of features aiming to characterize the effectiveness of software teams. These features are used to compose a set of metrics, *STEM*, for measuring five dimensions of software team effectiveness.
- We propose a machine learning based method that utilizes the proposed features to recommend software teams, *WESTREC*, given project description and a project leader.
- We perform a comparative study on state-of-the-art approaches for recommending software teams on the two well-known open-source software systems.
- We make the datasets and source code available online for research purposes at: *github.com/NoppadolAssava/STEM*.

The rest of the paper is organized as follows: Sect. 2 describes the background knowledge of collaborative software development and the algorithm to recommend teams for software development. The methodology of the WESTREC approach and the STEM are explained in Sect. 3. Section 4 discusses the results from the evaluation of WESTREC. Section 5 discusses the effectiveness of the actual teams formed by the project assignee. Section 6 discusses the threats to validity of our research. Section 7 reviews related literature. Section 8 concludes this paper.

2 Background

This section provides background concepts and knowledge related to collaborative software development, including the software development process and collaborative software team on the well-known open-source software tracking system, Jira.

2.1 Software development process

There are various types of software development life cycles in software engineering, such as the Waterfall, Spiral, Iterative and Incremental, Agile models, etc. Even though different software development methodologies have distinct processes, structures, and flows, all must include four activities fundamental to software engineering (Sommerville 2011).

1. **Software Specification**: The functionalities of the software and constraints on its operation must be defined based on customer needs.
2. **Software Design and Implementation**: Production of the software to meet the specifications.
3. **Software Validation**: The software must be validated to ensure that it meets the customers' needs.
4. **Software Evolution**: Improvement of the software to meet changing customer needs.

These four activities are included in every software development process and are part of the life cycle. The software development life cycle comprises six steps: planning, analysis, design, development, implementation, testing, and maintenance. Most software trackers, such as Jira issue tracking systems, keep track of these work-progressing items during the project execution.

2.2 Open source software team collaboration

The software industry has been successively developed due to the changes in technology and business. As the business moves faster than before, the requirements, systems, and entire business have a tendency to change rapidly. Many software project managers have attempted to improve the software development process with various management models and techniques. A new software development paradigm has been introduced in the last decade, namely the Open Source Software (OSS) paradigm. The licensing terms of OSS make its source code publicly available and allow redistribution of modified versions. OSS allows collaborative development by geographically dispersed team members who can work remotely. The key factor in a successful OSS is its organization with a different collaboration structure (Colazo 2010). Lack of communication could decrease the level of trust and frequent feedback, resulting in a lack of coordination and control over the software project (Khan et al. 2017a, b; Niazi et al. 2016). Furthermore, one of the critical success factors for global software development is progress measurement (Akbar et al. 2019). One of the tools that help to work on OSS or develop software with a complex process is project management software, which manages and supports project activities such as planning, monitoring, and controlling. Various software management systems are available in the market to facilitate the collaboration of these OSS practitioners. A prevalent example of such systems is the Jira[3] software tracking platform.

2.3 Jira: a software development tracking system

Developed and maintained by Atlassian since 2002, Jira is a powerful tool for project management and is typically used for tracking issues. A unit of work is referred to as an issue or task in Jira. Jira allows the user to plan, track, and

[3] https://www.atlassian.com/software/jira.

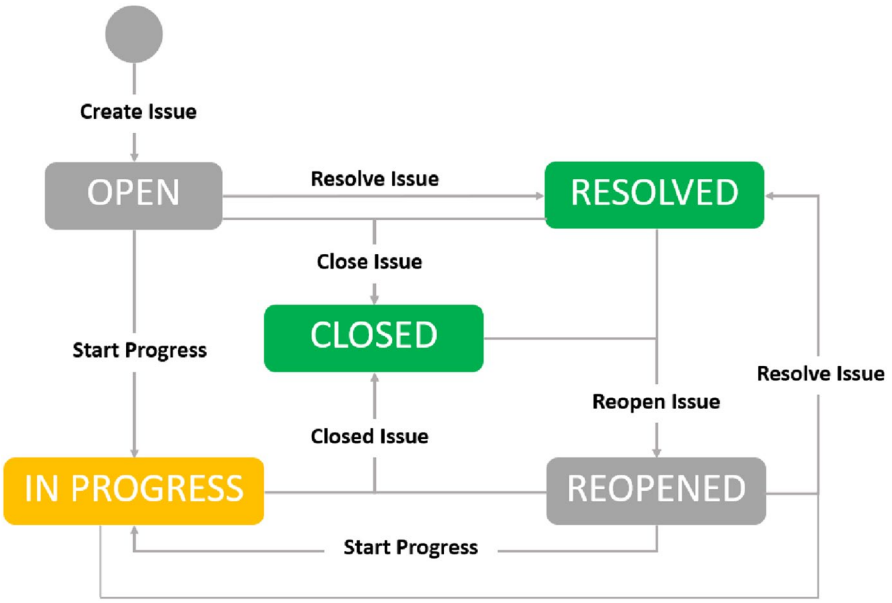

Fig. 1 The workflows in Jira (iDalko 2018)

release software. When the user wants to create an issue, the user must determine the issue type, depending on the configuration set by the project leader. The issues can be classified into various types, such as bugs, improvements, new features, and tasks. To facilitate tracking of collaborative software development activities, Jira has features that allow team members to easily track the work progress of software development. Jira has become a popular tool not only because of its ability to customize the system and make it more suitable to conduct the process in an organization but also because it can integrate with other tools such as Bitbucket, Git, and Salesforce. It also provides three main features to support the agile practice for all team members throughout the execution of the project, including planning the development iteration, creating iteration reports, and tracking bug functionalities (Sokolov 2017).

2.3.1 Issue types

Issues are building blocks of a Jira project. An issue could represent a story, a bug, a task, or another type of issue in a project. Different types of issues in Jira include:

- Bug: A bug is a problem that prevents the functions of a product from working correctly.
- Epic: An epic is a core user narrative that needs to be broken down into stories.
- Story: A user story is the smallest unit of work that needs to be done.

- Subtask: A subtask is a piece of work that is required to be done to complete the main task.
- Task: A task represents a unit of work that needs to be done.

2.3.2 Workflow

Figure 1 shows the issue status transition in the Jira platform, where an issue can be in one of the five states: open, in-progress, closed, reopened, and resolved. During the project execution, the workflow starts when a reporter (e.g., project leader, team member, user) creates an issue whose status is initialized to *Open*. When a developer starts working on the issue, the status is changed to *In Progress*. When the task is finished, the assignee can change the issue status to *Closed* to close the task. However, a closed issue (i.e., a resolved issue) can be *Reopened* if the assignee finds that the issue was not completely resolved. The *Resolved* status is given to signify that an issue is successfully resolved.

2.3.3 Team roles

Since Jira focuses on the implementation aspect of software projects, all the roles of team members are then associated with the development phase, including:

- **Reporter**'s main responsibility is to log the bugs. This role is automatically assigned by the tracker.
- **Assignee** is the one opening an issue and committed to complete the task. In this paper, an assignee is treated as a project leader.
- **Developer** can edit issues and assign issues to themselves. Typically, a developer is involved in the coding parts of the project.
- **Tester** validates the solution at the functional level, according to the test instructions provided.
- **Peer Reviewer** checks fixes or modifications at the code level.
- **Integrator** integrates different pieces of the solution into the codebase.

However, we note that a Jira team does not necessarily comprise all the above roles. It depends on the team's issue specification, complexity, and size.

2.4 Application of potential game theory in recommendation systems

A Potential Game consists of a set of players and an action set for each player with the assumption that there is a single global function that expresses the incentive of all players, namely the potential function (Monderer and Shapley 1996). Each player can choose any of his actions to increase the incentive, i.e., players try to optimize the potential function. In every potential game, at least one Nash equilibrium is the local optima of the potential function where a non-player can increase the potential function by changing his action alone. This can be used for team recommendation

problems by considering the roles in the team as players and the set of candidates for each role as the action set for each player while considering the team scoring function as the potential function (Liu et al. 2014). In this research, Max-Logit (Song et al. 2011), one of the Nash Equilibrium finding algorithms, is adopted. When comparing Max-Logit to other Nash equilibria algorithms, it can converge to the approximately best Nash equilibrium with provably the fastest convergence rate (Liu et al. 2014). The Max-Logit algorithm for team recommendations contains two major steps, including constructing a new team and deciding whether to accept it as the new best team or not (Liu et al. 2014).

It is important to note that the Max-Logit algorithm is primarily used for the purpose of accelerating the search for optimal team configurations. This is in contrast to a brute-force method, which involves enumerating and ranking all of the possible team configurations, which could be exponentially more expensive to compute. While other optimization algorithms, such as the Differential Evolution (Storn and Price 1997) and Non-dominated Sorting Genetic (Chen et al. 2017) algorithms, experimenting with different Nash-equilibrium finding algorithms is not in the scope of our research. Therefore, we leave such exploration for future work.

3 Methodology

We first discuss the overview of our proposed software team recommendation algorithm, WESTREC, which covers the proposed software team features, team effectiveness score, and team recommendation. Then, we discuss the proposed software team effectiveness metrics, STEM.

A user is defined as a software practitioner who has an account in the target software tracking platform (e.g., Jira). A software team then comprises a set of users, each referred to as a team member. Let $R = \{r_1, r_2, r_3, ...\}$ be the set of team roles and $U = \{u_1, u_2, u_3, ...\}$ be a set of candidate users. Here $U_r \subseteq U$ is the set of users compatible with role $r \in R$. A team $T = \{< r_1, u_1 >, < r_2, u_2 >, ...\}$ is a set of tuples, each of which comprises a role and its corresponding user. Each project p consists of an assignee u_0 who is responsible for assigning a job to other roles, a set of team members T, and a project description d. Hence, a project can be represented as $p =< u_0, T, d >$, where $u_0 \in U$ and all of members in $T \in U$. *WESTREC* returns a ranked list of K potential software teams, sorted by Team Effectiveness score (TES). That is $\text{WESTREC}(u^*, d^*, r^*) = \{T_1, T_2, ..., T_K\}$, where $\text{TES}(T_i) \geqslant \text{TES}(T_j), i < j$, for a given project assignee u^*, project description d^*, and a set of roles required for the project r^*. Besides considering relevant experience, *WESTREC* also takes into account the relationship and correctness among the team members expressed in the collaboration network and the compatibility of skills of team members to ensure that the recommended teams are suitable for the given project.

Figure 2 illustrates the overview of *WESTREC*. *WESTREC* uses the historical software development processes as the training data for recommending compatible team configurations for a given project. Therefore, it first collects the project's development information from the Jira (Sect. 3.1). Then, the data is prepossessed (i.e., removing incomplete projects), after which the information

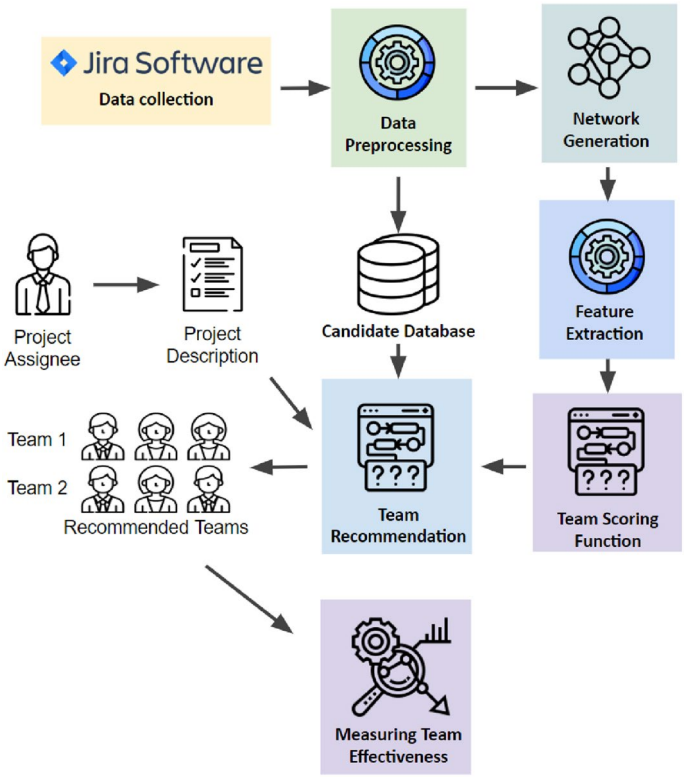

Fig. 2 High-level diagram of WESTREC

required for *WESTREC* (e.g., candidate software practitioners for each role, project description, etc.) is extracted, described in Sect. 3.2. *WESTREC* then constructs knowledge graphs (e.g., User Interaction Network, Project Dependency Network, etc.) which will be described in Sect. 3.3. Such graphs allow *WESTREC* to use collaboration history, skills, and relevant experience for feature extraction. The various features are extracted from the knowledge graphs, characterizing important aspects of an effective software team with respect to a given project, such as skill compatibility of team members, collaboration among team members, and compatibility of team members toward the project assignee (Sect. 3.4). As *WESTREC* needs to rank teams as part of the recommendation mechanism, the metrics to represent the team effectiveness (i.e., Team Effectiveness Score) to a given project are introduced. Such scores can be calculated by applying machine learning classification models to approximate the team's effectiveness by leveraging the project's outcome (i.e., winning, non-winning), which is described in Sect. 3.5. Then, the team effectiveness scoring function is used in the team recommendation framework that utilizes a variant of Max-Logit to approximate teams with the highest Team Effectiveness scores and

outputs the top K teams (Sect. 3.6). In the final step, the recommended teams are evaluated based on several metrics, namely our proposed Software Team Effectiveness Metrics (STEM) (Sect. 3.7), and the ability to predict previous successful teams.

3.1 Data collection

Information related to software development projects is collected from the Jira software tracking platform due to its popularity among open-source software systems. Furthermore, such a platform provides useful information such as project descriptions, team members with explicit roles, change logs of the software development process, etc. Data from Jira can be collected through the Jira REST API, which provides the data in JSON format. Examples of the software systems that use Jira to track their projects include Apache[4] and Atlassian[5].

3.2 Data preprocessing and extraction

The collected JSON files contain a huge amount of information, both related and non-related to our research. Hence, we only extract information minimally required to operate *WESTREC*, to not only minimize possible assumptions about software projects and teams but also allow the proposed method to generalize across multiple software tracking platforms.

3.2.1 Team members and roles

In this research, we refer to each software development unit in Jira (i.e., bug, epic, story, task, etc.) as a *project* for simplification and generalization across multiple software tracking platforms. A project must contain an assignee who is responsible for selecting the team members and assigning appropriate roles for them. In our research, we refer to the project assignee as the *project leader* to align with the global notation of a collaborative software project (Franzago et al. 2017). Besides the project leader's role, we also collect essential roles: developer, tester, peer reviewer, and integrator. While only these roles are used to establish case studies in this research, *WESTREC* is designed to allow the addition of new roles.

3.2.2 Project status and project resolution

Project status in Jira can be either *open*, *in progress*, or *closed*. We only retain *closed* projects since the project outcomes are needed to label training data. Furthermore, each project has a resolution label that tell how the project is closed, i.e. *Resolved*, *Deferred*, *Incomplete*, *Fixed*, *Done*, etc. Since *WESTREC* aims to recommend

[4] https://issues.apache.org/jira.
[5] https://jira.atlassian.com/secure.

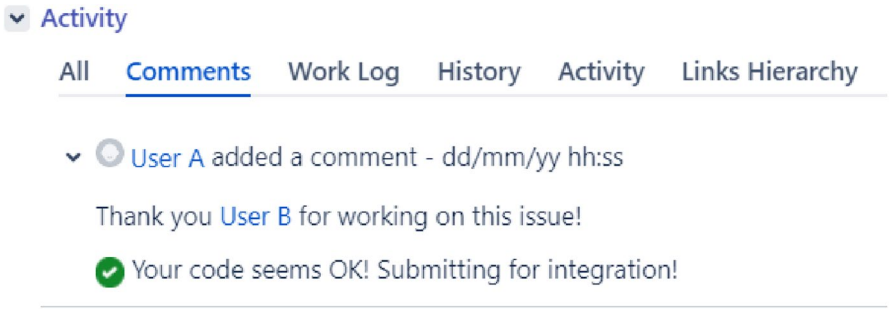

Fig. 3 User communication through a comment in the Moodle tracker

effective software teams with a higher chance of completing given projects, we further classify projects into "winning" and "non-winning" projects, so that machine learning algorithms can learn to characterize effective teams from the winning projects. Technically, a winning project must satisfy the following criteria:

1. The project status is *Closed*.
2. The project resolution label is either *Resolved*, *Fixed*, or *Done*. Note that a project with a *Deferred* label indicates a delay; hence, more time is needed and may imply ineffective planning or slow team progress. Similarly, a project with an *Incomplete* label may also indicate unsatisfactory achievement. In both cases, these deferred and incomplete projects are classified as non-winning.
3. The project has never been *reopened*. This is because research has shown that one of the major causes of reopened projects stems from ineffective teams (Assavakamhaenghan et al. 2019).

Projects that do not fit all the criteria above are labeled as *non-winning*. Such a binary project labeling scheme also encourages the proposed method to recommend teams with characteristics similar to the winning teams. Note that the criteria for labeling winning teams ensure the high quality of the positive samples. It does not mean that non-winning projects are unsuccessful ones especially due to ineffective teams, since there can be many factors not-related to team composition that prevents these teams from satisfying the winning-team criteria, such as project scope change, project termination, or simply reusing existing projects as place-holders (hence reopening them).

3.2.3 User interaction

Healthy interaction among team members is key to a successful project. Lack of communication could decrease the level of trust and frequent feedback, resulting in a lack of coordination and control over the software project (Khan et al. 2017a, b; Niazi et al. 2016). In Jira, interaction can be extracted from tagged users in the comments. For example, in Fig. 3, User A expresses appreciation to User B for integrating the solution. Furthermore, sentiment analysis of comments can indicate the

relationship among team members. We use SentiStrength[6], a python library for sentiment analysis, to quantify the sentiment polarity from each comment. The value of the comment sentiment can be between [-5, 5], where -5/5 indicates extreme negative/positive sentiment.

According to Kale (2007), trust and distrust can be propagated from one person to another. Borrowing the similar concept, link polarity can be propagated using the following matrix operation:

$$C = \alpha_1 B + \alpha_2 B^T B + \alpha_3 B^T + \alpha_4 B B^T \tag{1}$$

$$B_{i+1} = B_i * C_i \tag{2}$$

Where C is the atomic propagation operator, B is the initial trust between user i and j. We experimented with several values for the α vector and discovered that the default $\alpha = \{0.4, 0.4, 0.1, 0.1\}$, as also found optimal by Guha et al. (2004) and Kale (2007), produced the best results. Using the atomic operator matrix, the trust can be computed iteratively depending on the steps of propagation $i + 1th$. B_{i+1} is calculated using Equation 2.

3.3 Knowledge graph generation

Since the software team data contain many historical relationships between entities (i.e., software practitioners, projects, and skills). The relationship between them can be very important for the recommendation of collaborative software teams. One of the intuitive ways to represent such relations is using graphs. Thus, many knowledge graphs are introduced in this research. This section describes each knowledge graph we establish.

3.3.1 User interaction network

The user interaction network is a person-to-person graph that determines the relationships between people in the directed graph format. Each node represents each person, and each edge has two weights representing the calculated positive and critical sentiment scores. The sentiment is derived from interactions between members that can be observed from tagging between members in comments (described in Sect. 3.2.3).

3.3.2 User collaboration network

The user collaboration network is a person-to-person graph that determines the relationships between people in an undirected graph. Each node represents a person, and the weight of the edge connected between each pair of nodes is derived from the number of projects they have collaborated on.

[6] http://sentistrength.wlv.ac.uk/.

3.3.3 Project dependency network

The project dependency network is a project-to-project graph determining the relationship between each project in a directed graph format. Each node represents each project, and an edge represents the relationship between each project, such as blocking, being cloned from, depending on. These relationship types indicate the relationship between projects. For example, the *clone* relation indicates that a project is cloned from another project.

3.3.4 Project similarity network

The project similarity network is a project-to-project graph that determines each project's relationship in an undirected graph. Each node represents each project, and the edge's weight represents the similarity between a pair of projects. The similarity score is calculated from the cosine similarity of the topic distributions of the pair of projects. The cosine similarity is one of the well-known metrics for vector similarity measurement that is wildly used in many software engineering papers (Xu et al. 2018; Maalej et al. 2017). A topic distribution is derived from applying Latent Dirichlet Allocation, which is a topic modeling algorithm, to the project description. In this research, the project description is the concatenation of the project's title, components, and description.

3.3.5 Skills network

The Skills Network is a skills-to-person graph that determines each person's skills in an undirected graph. In this research, skills are referred to as the components a project belongs to (e.g., Javascript, Interface, etc.). The Nodes represent the people and their skills. Lastly, the weight of each edge $u - s$ represents the number of projects related to component s that user u has participated in.

3.4 Feature extraction

Features that characterize team effectiveness are extracted from a given team with respect to the given project description and project leader. In addition to our proposed features, the features proposed by Liu et al. Liu et al. (2014) are also adopted. Note that all features are normalized with the Min-Max normalization. To calculate the overall value of the team from each member's value, the average is used.

3.4.1 Features adopted from Liu et al. (2014)

1. **Experience**: The number of projects (i.e., tasks) that a person has been involved in.
2. **Role Experience**: The number of projects in which a person participated in a particular role.

3. **Win Experience**: This feature is the number of successfully resolved projects that a person participated in.
4. **Win Rate**: This feature is the ratio of `Win Experience` to `Experience`.
5. **Closeness**: This feature is calculated using the following equation (Liu et al. 2014).

$$Closeness = \frac{2}{|T| \times (|T| - 1)} \sum_{u_i, u_j \in T} \frac{1}{SP(u_i, u_j)} \tag{3}$$

$|T|$ is the cardinality of the team. $SP(u_i, u_j)$ is the shortest path from u_i to u_j. The *ShortestPath* is calculated from the graph $G = (U, E)$ where U is the set of users. There is an edge $e_i \in E$ linking $u_i \in U$ and $u_j \in U$ with a weight of 1 if u_i and u_j have worked together on a same project. If there is no path between u_i and u_j, $|U|$ is used instead.

6. **Connection**: Liu et al. (2014) described this feature as follows.

$$Connection = \frac{2}{|T| \times (|T| - 1)} \sum_{u_i, u_j \in T} e_{ij} \tag{4}$$

Where e_{ij} is the number of connections (represented as tags in comments) between u_i and u_j.

3.4.2 Our proposed features

1. **Project Familiarity**: *Project Familiarity* reflects a team experience gained from dealing with similar projects in the past. It is derived from the Project-Project similarity network. Each project is represented as a vector of topics inferred from the topic modeling model using LDA (Blei et al. 2003). Then, the given project is compared with experienced projects of team members using cosine to find the similarities. The *Project Familiarity* is calculated by looking at each member's average max project similarity.

$$ProjectFamiliarity = \frac{1}{T} \sum_{u \in T} max_{I \in P_u}(cosine(I^*, I)) \tag{5}$$

Here T is a set of team members, p is a project, I is a project represented in a vector of topic distribution, I^* is the inputted project, and P_u is the set of projects that user u participated in.

2. **Component Experience**: *Component Experience* quantifies the expertise of a team related to a required component, computed by the number of projects in which each team member participated in that are related to the required components.

$$ComponentExperience = \frac{1}{|T||C_r|} \sum_{u_i \in T} \sum_{c_j \in C_{u_i} \cap C_r} f(u_i, c_j) \tag{6}$$

Here, T is a set of team members, u is a user, C_r is the set of required components and C_u is the set of components that user u has experienced, and $f(u, c)$ is the number of projects that user u experienced which are related to component c.

3. **Project Domain Experience**: The software tracker platform comprises multiple divisions, i.e., project domains, that focus on different software components or functions. Hence, *Project Domain Experience* can reflect the team members' experience while working on a specific project domain. This feature is derived from the average number of projects that team members have worked on in a particular domain.

$$ProjectDomainExperience = \frac{1}{|T|} \sum_{p \in T} f(u, p) \tag{7}$$

Here, T is a set of team members, u is a user, p is the input project and $f(u, p)$ is the number of projects in project p that user u resolves.

4. **Team Interaction (Team Positivity and Team Criticism)**: The *Team Interaction* scores reflect how members of a team interact, derived from tags and comment sentiment analysis when members address each other regarding a project. A trust propagation method (Kale 2007) is adopted to expand the relationship between pairs of users. Both positive and critical (negative) sentiment values of comments are considered for the interaction scores. The *Team Interaction* scores are the average interaction between each pair in a team, where the Positivity Team Interaction score is calculated from the positive interaction and the Criticism Team Interaction from the criticism interaction.

$$TeamInteraction = \frac{1}{|T| \times (|T| - 1)} \sum_{u_i, u_j \in T, i \neq j} score(u_i, u_j) \tag{8}$$

Here, T is a set of team members, u is a user, $score(u_i, u_j)$ is interaction score between user u_i and u_j, derived from the User Interaction network. Note that $score(u_i, u_j)$ utilizes positive sentiment when calculating the Team Positivity feature and criticism sentiment when calculating the Team Criticism feature.

5. **Interaction Scores toward Assignee** ($TeamPositivity_{u_A}$ **and** $TeamCriticism_{u_A}$): Similar to the *Team Interaction* features, the *Interaction Scores toward Assignee* features comprise two interaction scores, namely positivity and criticism. These features focus on how an assignee interacts with the whole team, reflected by the sentiment from the team members towards the assignee.

$$TeamInteraction_{u_A} = \frac{1}{|T|} \sum_{u_i \in T} score(u_A, u_i) \tag{9}$$

Here T is a set of team members, u is a user, u_A is the assignee, and $score(u_A, u_j)$ is the interaction score for the interactions between assignee u_A and u_i.

6. **Team Contribution**: *Team Contribution* reflects how often a person contributes through an activity to deal with a project, such as changing the project status

or updating code. A person's contribution is derived from the average ratio of a person's activities in the previous projects.

$$TeamContribution = \frac{1}{|T|} \sum_{u \in T} \frac{\sum_{p \subset P_u} c_{u,p}}{|P_u|} \tag{10}$$

Here T is a set of team members, u is a user, P_u is the set of projects that user u participated in, $c_{u,p}$ is the contribution of a user u towards project p which can be calculated by the following formula:

$$c_{u,p} = \frac{|Activities\ of\ u\ in\ project\ p|}{|Activities\ in\ project\ p|} \tag{11}$$

7. **Co-Project Frequency**: *Co-Project Frequency* reflects how often the team members work together in the same previous projects. It is derived from the average number of times a pair of members work together.

$$Co - ProjectFrequency = \frac{1}{|T| \times (|T| - 1)} \sum_{u_i, u_j \in T, i \neq j} W_{u_i, u_j} \tag{12}$$

Here, T is a set of team members, u is a user, and W_{u_i,u_j} is the number of times u_i and u_j work together.

8. **Skills Competency**: The *Skills Competency* reflects how the skills of a team are compatible when dealing with a project. It considers the skills of team members and the required skills to deal with a project. The components of a project are considered skills. It is measured by the number of team members' skills that match the required skills divided by the number of required skills.

$$Skills\ Competency = \frac{|S_T \cap S_r|}{|S_r|} \tag{13}$$

Here S_T is the set of team members' skills, S_r is the set of required skills to deal with a project.

9. **Skills Diversity**: *Skills Diversity* reflects the skill diversity of the team members. The components of a project are considered skills. The more variety in the components, the higher the *Skills Diversity* score. It is measured by the number of unique required skills of a team divided by the team members' total required skills.

$$Skills\ Diversity = \frac{|S_T \cap S_r|}{\sum_{u_i \in T} |S_{u_i} \cap S_r|} \tag{14}$$

Here, T is the set of team members, S_T is the set of team members' skills, S_r is the set of required skills to deal with a project, and S_{u_i} stands for the skills of the person u_i.

10. **Project Proximity** (between Assignee and teams): *Project Proximity* reflects the distance between the members in historical work aspects, derived from the

Issue-Issue network. The minimum of the shortest paths between the projects that the pair of team members resolved is calculated. *Project Proximity* is the inverted average of the values from all pairs of team members. The minimum of shortest paths refers to how close the people are when considering the project distance. This feature considers only pairs of the assignee to other team members. This is because, in some cases, a person may be involved in more than 1,000 projects, resulting in a huge number of shortest path calculations, making it extremely computationally expensive.

$$ProjectProximity = \frac{|T| \times (|T| - 1)}{\alpha + \sum_{u_i \in T} min_{p_i \in P_{u_A}, p_j \in P_{u_i}} (SP_N(p_i, p_j))} \tag{15}$$

Here, T is a set of team members, u is a user, u_A is the assignee, p is a project, and P_u is the set of projects p that a user u resolves. The $SP_N(p_i, p_j)$ is the shortest path between p_i and p_j of which the maximum length is N, and α is the smoothing factor to prevent a zero division. In this research, we used $\alpha = 1$ and $N = 1,000$.

11. **Team Coherence**: *Team Coherence* reflects how close the team is by considering the distance between people derived from the User Collaboration Network. It is calculated by taking the inversion of the averages of the shortest paths between each pair of people. The longer the path, the lower the relatedness. We consider the relatedness of the whole team through the following formula.

$$TeamCoherence = \frac{|T| \times (|T| - 1)}{\alpha + \sum_{u_i, u_j \in T, i \neq j} (SP_N(u_i, u_j))} \tag{16}$$

Here, T is a set of team members, u is a user, $SP_N(u_i, u_j)$ is the shortest path from u_i to u_j with a maximum length of N, and α is the smoothing factor to prevent a zero division. In this research, we use $\alpha = 1$ and $N = 1,000$.

12. **Relatedness toward Assignee** ($Relatedness_{u_A}$): This feature is similar to the Team Coherence feature. This feature focuses on how the assignee relates to the whole team.

$$Relatedness_{u_A} = \frac{|T| \times (|T| - 1)}{\alpha + \sum_{u_i \in T} (SP_N(u_A, u_i))} \tag{17}$$

Here, T is the set of team members, p is a user, $SP_N(u_A, u_j)$ is the shortest path from assignee A to u_j with a maximum of N, and α is the smoothing factor to prevent a zero division. In this research, we use $\alpha = 1$ and $N = 1,000$.

3.5 Team scoring function

The *Team Effectiveness* score (TES) refers to the compatibility of a team to successfully complete a particular project. A machine-learning algorithm is used to learn from the historical data (i.e., historical project outcomes for several team features) and estimate the team's chance to successfully resolve a given project. In this

research, the tasks in the training datasets are labeled as *Winning* and *Non-Winning* based on the criteria in Sect. 3.2.2. However, the data is very skew. The number of winning projects is approximately 11 and 18 times the number of non-winning projects in Apache and Atlassian, respectively. Thus, we mitigate this problem by generating 100 additional *non-winning* teams for each project. Each generated not-winning team is produced by starting with a real winning team and then replacing its team members with other random users. Each newly generated team must be distinct from the previously established *winning* teams. The creation of additional negative samples (not-winning teams) helps to balance the dataset. This, in turn, enables the machine learning model to represent the qualities possessed by the winning teams more accurately. A possible effect from adding these synthetic trivial negative samples could result in better performance than training the models with only actual team data. However, since all the models are trained and tested with the same datasets, their performances are comparable on the same ground.

To select the best machine learning algorithm for *Team Effectiveness* scoring function, various algorithms drawn from diverse families of machine learning algorithms are investigated, including Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Naive Bayes (NB), Quadratic Discriminant Analysis (QDA), k-Nearest Neighbors (kNN), and Neural Network (NN). These are traditional machine learning techniques that are wildly used in many reputable papers (Liu et al. 2014; Choetkiertikul et al. 2017). Then, the 5-fold cross-validation is performed to select the best machine learning algorithm. Because we aim to use the raw probability generated by the model in the downstream tasks rather than the final classification results, the Area Under the Receiver Operating Characteristic curve (AUROC) is used to select the optimal machine learning method. Therefore, other classification evaluation metrics such as precision, recall, and F1 are utterly irrelevant because they take the cutoff probability into account. As a result, AUROC is employed as the criterion for selecting the best machine-learning algorithm to compute *Team Effective* scores.

3.6 Team recommendation

While it is intuitive to enumerate all the possible team combinations, compute TES for each of them, and recommend the top teams with the highest TES, exhaustedly enumerating all team possibilities can be computationally expensive, especially in large software systems with thousands of candidates for each role. Thus, Max-Logit (Song et al. 2011) is adopted to approximate the best teams by pruning out subsets of possible team configurations that are expected to have low TES. Furthermore, we modified the original Max-Logit algorithm to track all the teams in each iteration and rank them based on the cost function at the end. The Cost(T) function is defined as the inverted Team Effectiveness score (i.e. $\frac{1}{\textit{Team Effectiveness Score}}$). Our modified Max-Logit algorithm is outlined in Algorithm 1.

input : A project with a set of required roles, set of candidates for each role, function $Cost(T)$ is the inverse of the TeamStrength score using features from T, number of iterations N, smoothing factor τ, and number of recommended team K
output: top K best teams and their cost

$RecTeam$ = array()
Randomly select candidate for each role and generate a team T
Append $(T, Cost(T))$ to $RecTeam$
for $i = 1$ **to** N **do**
    Calculate $Cost(T)$
    Randomly select a role, and replace it with a randomly selected an alternative candidate, get a new team T'
    Calculate $Cost(T')$
    Append $(T', Cost(T'))$ to $RecTeam$
    $prob \leftarrow PROBABILITY(Cost(T), Cost(T'))$
    $r \leftarrow random(0, 1)$
    if $r \leq prob$ **then**
        $T \leftarrow T'$
    end
end
sort $RecTeam$ on $cost$ ascending
$RecTeam \leftarrow RecTeam[: K]$
return $RecTeam$

Function $PROBABILITY(Cost(T), Cost(T'))$
    $v_t = exp^{-Cost(T)/\tau}$
    $v_{t'} = exp^{-Cost(T')/\tau}$
    $prob = \frac{v_{T'}}{max(v_t, v_{T'})}$
    return $prob$
end

Algorithm 1: Maxlogit for the topK Best Team

3.7 Software team effectiveness metrics (STEM)

Compared to the traditional method of manually selecting team members, having the capability to automatically propose software team configurations not only could help those responsible for assigning tasks to save time but also could help them establish a more efficient team. For instance, a task on implementing a chatbot functionality to the Moodle system could require the following individuals: a data scientist who collaborates with domain experts to understand the requirement and experiment with a variety of models; a machine learning engineer who designs the ML pipeline and implements the chatbot services; a web programmer who develops the front-end web interface; an integrator who connects different components via the APIs; and a tester who validates the expected functionality of the chatbot. In the absence of an intelligent system that provides team recommendations, an assignee would have to look for previous relevant tasks to whittle down possible applicants for the needed roles and find suitable team members. Then, the task assignee is responsible for attempting to evaluate the candidates' qualifications and getting in touch with them. In addition, given that the assignee

selects every single member of the team on their own, it is possible that they will not work well together in practice since they may not have worked together before or had conflicts in previous tasks. This is especially true when members of the team do not feel comfortable communicating directly with one another and offering both positive comments and constructive criticism, resulting in high communication costs and poor morale within the team. Therefore, a preferred algorithm for recommending teams should not only strive to improve the skill component of the collaborative effort but should also seek to optimize the teamwork compatibility aspect as well. In this section, we propose STEM (Software Team Effectiveness Metrics), aiming to quantitatively measure the effectiveness of a software team in various aspects, including communication cost, team morale, cohesiveness, experience, and skills.

3.7.1 Communication cost

One of the most common team effectiveness metrics widely used in previous literature is communication cost (Wang et al. 2016), typically defined as the closeness between experts in the social graph. If an edge connects two experts, the edge weight is used as the communication cost. For a pair of non-adjacent experts, the communication cost can be calculated by finding the weighted shortest path between the two experts. To align with the previous work, in our setting, users could be treated as experts. We calculate the communication cost from the User Collaboration network with the weights replacement of 1. To formulate the effective communication cost function for a team, Wang et al. (2016) introduced the following formulations:

- **CcR** represents a radius distance derived from the longest shortest path between team members.
- **CcSteiner** uses the Steiner distance, derived from the weighted cost of the minimum spanning tree taken from the subgraph within the team.
- **CcSD** is the sum of the distance in a team, derived from the sum of all the shortest paths between any two team members.
- **CcLD** is the communication cost from the members to the leader, derived from the sum of the shortest path from the team leader (assignee) to each of the team members.

The purpose of these functions is to calculate the value that can represent the overall communication cost of a team. A previous study has shown that the communication cost can be an indicator that quantifies the effectiveness of team collaboration (Lappas et al. 2009). The lower the communication cost means, the closer the team members are, thus resulting in productive collaboration (Storey et al. 2017). The total communication cost (CC) represents the overall communication cost of a team, defined as:

$$CC = \frac{CcR + CcSteiner + CcSD + CcLD}{4} \tag{18}$$

Using communication cost functions alone may not be sufficient to measure the effectiveness of a software team since they only consider the closeness between

team members. Therefore, we propose additional metrics to reflect more complete aspects of an effective software team. The proposed metrics are derived from the findings from state-of-the-art software team effectiveness models.

3.7.2 Team morale

Team morale (TM), or team coordination, is one of the factors that affect team productivity and success (Dingsøyr et al. 2016). Team morale statistics are computed from interaction amongst team members (Dingsøyr et al. 2016). Many studies show that higher quality of interaction and morale correlate with the project success (Chipulu et al. 2014; Alsharo et al. 2016). Therefore, the proposed metrics that indicate team morale include **Team Interaction** scores, **Interaction Scores toward Assignee**, and **Connection**. These interaction-based features represent the sentiment of the interaction (i.e., through comments) between members of the team. Praising messages (positive sentiment) could lift the collaboration willingness among team members (Grigore and Rosenkranz 2011), while criticism messages on project functionality (criticism sentiment) could lead to an improvement in the quality of the project (Wieland et al. 2013). The **connection** feature shows how frequently team members communicate with each other. Hence, this feature can be an indicator of team coordination. To reflect the overall team morale, the *TM* metric is defined as:

$$\begin{aligned} TM = \frac{1}{5} \cdot (&TeamPositivity + TeamCriticism \\ &+ TeamPositivity_{u_A} + TeamCriticism_{u_A} \\ &+ Connection) \end{aligned} \tag{19}$$

3.7.3 Team cohesion

Another factor that positively affects the team's effectiveness is cohesion (Dingsøyr et al. 2016). Team cohesion (TC) is defined as "the tendency for a group to stick together and remain united in the pursuit of its goals and objectives" (Mudrack 1989). Lack of group cohesiveness among distributed team members results in a weakening of communication which causes diverse communication issues (Khan et al. 2014). In our case, the goal and objective of a team are to complete the given project. The cohesion can be expressed by the combination of the following WESTREC features: **Co-Project Frequency**, **Project Proximity**, **Team Cohesiveness**, **Relatedness toward Assignee**, and **Closeness**. Co-Project Frequency, Team Cohesiveness, and Relatedness to Assignee indicate how well team members have adhered to one another in past projects. These features focus on counting the frequency and measuring the closeness in the User Collaboration network. Furthermore, collaboration frequency among team members was found to be an important factor that positively affects team performance (Huckman et al. 2009; Lindsjørn et al. 2016). The higher of these features show the team's higher cohesion (i.e., nodes stick together). Project Proximity shows cohesion in the aspect of the project dependencies (i.e., using the Project Dependency network). A high Project

Proximity value indicates that the team members have worked together on related projects, resulting in high team cohesion. The overall team cohesion (TC) value is computed by averaging all the related metrics:

$$\begin{aligned} TC = \frac{1}{5} \cdot (&CoProjectFrequency + ProjectProximity \\ &+ TeamCoherence + Relatedness_{u_A} \\ &+ Closeness) \end{aligned} \tag{20}$$

3.7.4 Team experience

Another important factor that strongly affects the team's effectiveness in solving a particular problem is the collective experience and expertise. Extensive literature found that experience and skill competency are crucial for software development (Beaver and Schiavone 2006; Huckman et al. 2009; Jiang and Klein 2000; Niazi et al. 2010). Furthermore, coordination of expertise is found to be a factor that affects the team performance (Faraj and Sproull 2000). Therefore, a team should contain a diversity of skills. From this knowledge, we propose to use the following features as metrics to quantify the experience aspect of a team: **Project Familiarity**, **Component Experience**, **Domain Experience**, **Team Contribution**, **Skill Competency**, **Skill Diversity**, **Win Rate**, and **Role Experience**. These metrics can be divided into two groups based on what they capture: Team Experience metrics and Team Proficiency metrics.

Team Experience (TE) metrics consist of **Project Familiarity**, **Component Experience**, **Domain Experience**, **Win Rate**, **Team Contribution**, **Experience**, and **Role Experience**. These metrics indicate the number of projects that team members have completed, which infer the collective experience of the team members. Furthermore, **Team Contribution** also shows how actively the team members participated in their previous projects. Furthermore, **Win Rate** shows how frequently they successfully complete their previous projects. Not only that the project domain experience positively affects the effectiveness, but the role experience also indicates how a member has relevant experience with respect to a particular role (Huckman et al. 2009). Hence, **Role Experience** is also included in the team experience metric. The overall team experience (TE) value is the average of all the related metrics:

$$\begin{aligned} TE = \frac{1}{7} \cdot (&ProjectFamiliarity + ComponentExperience \\ &+ ProjectDomainExperience + TeamContribution \\ &+ Experience + WinRate + RoleExperience) \end{aligned} \tag{21}$$

3.7.5 Team proficiency

The Team Proficiency (TP) metrics consist of **Skill Competency** and **Skill Diversity**. These metrics reflect whether the collective skills of the team members are

suitable for the given tasks. It is important that the team members possess not only sufficient but also diverse technical skills to avoid skill redundancy (Jiang and Klein 2000). **Skill Competency** indicates the coverage of the expertise with respect to the given project. In addition, **Skill Diversity** captures the variety of the technical skills of the team members (Faraj and Sproull 2000). The overall team proficiency metric can be computed as the average of the two atomic metrics:

$$TP = \frac{SkillCompetency + SkillDiversity}{2} \tag{22}$$

4 Evaluation of WESTREC

This section describes the evaluation protocols and their results. We address the following research questions:

RQ1: What is the best suitable machine learning model for the Team Effectiveness scoring function?

RQ2: Does WESTREC recommend teams that have the potential to resolve a project?

RQ3: How effective are the teams recommended by WESTREC compared to those recommended by other baseline approaches (i.e., Recency-based and Liu et al. (2014)'s approaches)?

The evaluation can be separated into two parts. The first part aims to find the most suitable machine learning model for the Team Effectiveness scoring function (**RQ1**). The second part focuses on quantifying the effectiveness of the recommended teams (**RQ2**, and **RQ3**).

4.1 Datasets

Our case studies selected two well-known open-source software systems (i.e., Apache and Atlassian) hosted on the Jira platform. The total numbers of projects in Apache and Atlassian are 507,319 and 238,322, respectively. After the data were collected through the Jira APIs, the data preprocessing and cleaning (i.e., filtering out unfinished and incomplete projects) were performed. The statistics of the datasets are shown in Table 1. Atlassian has a significantly lower proportion of winning projects (1.2%). This is because most of Atlassian's projects do not provide explicit roles for all the team members; hence they are treated as incomplete and filtered out from our useable datasets. On the other hand, the roles in most of Apache's projects (e.g., XalanJ2, Cassandra, etc.) are explicitly labeled. Finally, the data is split into 80% training and 20% testing sets based on projects' dates, where all the projects in the test are more recent than those in the training set.

Table 1 Statistics of the datasets

	Apache	Atlassian
#Completed Projects	43,196	2917
#Winning Projects[a]	39,440	2756
#Non-Winning Projects[a]	3756	161
#Component	782	170
#Developer	2265	39
#Tester	–	21
#Reviewer	41	127
#Integrator	–	–

[a]Winning and non-winning projects are defined in Sect. 3.2.2

Table 2 AUROC of each model among three platforms for team scoring function

Classification Model	Apache	Atlassian
Logistic Regression	0.979 ± 0.015	0.962 ± 0.022
Decision Tree	0.990 ± 0.002	0.774 ± 0.096
Random Forest	0.995 ± 0.000	0.980 ± 0.009
Naive Bayes	0.961 ± 0.017	0.939 ± 0.048
QDA	0.966 ± 0.016	0.936 ± 0.057
K-Nearest Neighbors	0.917 ± 0.016	0.811 ± 0.103
Neural Network	0.960 ± 0.021	0.939 ± 0.048

4.2 Scoring function evaluation

Binary machine learning classification algorithms are used to model the team effectiveness, which outputs the probability score ranging between [0,1]. A probability score close to 1 means the model predicts that the team will likely successfully complete a given project (i.e., a winning team). Each classifier is trained to recognize the winning teams using the features extracted from the winning and not-winning teams as discussed in Sect. 3.5. Different classification algorithms have their own mathematical models and architectures to compute the probability, which is used as the team score for the downstream tasks. To answer **RQ1**, a number of machine learning classification algorithms are validated, where the one with the highest AUROC is selected. The AUROC (area under the receiver operating characteristic curve) is used to represent the overall ability to describe the labeled data, regardless of the binary probability threshold used in other classification metrics such as precision, recall, and F1. Table 2 illustrates the average AUROC of each model using the 5-fold cross-validation protocol on the 80% training data. It is evident that Random Forest provides the highest AUROC among the two datasets. Therefore, Random Forest is chosen for the TES function for the subsequent experiments.

RQ1: Random Forest is founded to be the most suitable machine learning algorithm to be used as the scoring function for our datasets (i.e. Atlassian and Apache) with the AUROC above 0.9, and higher than those of other algorithms.

4.3 Team effectiveness evaluation

The teams recommended by WESTREC and the baseline approaches are compared in terms of the ability to predict previously winning teams. Each dataset is split into 80:20 portions based on the recency of the project dates. The older 80% of data is used for knowledge graph generation (Sect. 3.3) and training the team scoring function (Sect. 3.5). The rest (more recent) of the data is used as the test data. Quantitative experiments are performed to answer **RQ2** and **RQ3**.

4.3.1 Baseline approaches

Two approaches are used as the baselines for software team recommendations. The first approach implements a heuristic method that ranks and recommends teams whose members are recently active. Specifically, it first computes individual recency for each candidate team, the length (days) from which each member was active in his/her most recent project, then recommends teams with the smallest summation of the recency. This approach aims to mimic the project assignee's decision-making process when he/she selects a team member who recently worked in a software system. Another baseline approach is from Liu et al. (2014). This approach uses the Max-Logit algorithm to find teams with the approximately highest score. They use Logistic Regression with non-negative feature coefficient constraints as the team scoring function. WESTREC is similar to Liu et al. (2014)'s approach in the sense that it also uses Max-Logit to approximate best teams; however, WESTREC incorporated additional software engineering-specific features that can capture characteristics of effective software teams (e.g., expertise, skill diversity, activeness, etc.).

4.3.2 Previous winning team retrieval

A way to determine if a team has the potential to resolve a project is to identify whether the team was able to successfully solve a previous project or not. Hence, we measure the ability of the proposed approaches in recommending previous winning teams to answer **RQ2**. The ability to retrieve previous winning teams is evaluated by standard evaluation metrics for recommendation systems, comprising Mean Reciprocal Rank (MRR), Mean Rank (MR), Mean Rank of Hit (Hit MR), Hit at 10, Hit at 100, and Mean Average Precision (MAP). In this work, a relevant result refers to a team that previously successfully completed a given project. The evaluation metrics are described below:

- **Mean Reciprocal Rank (MRR)** measures the ability of the model to predict the first correct result, taking its rank into account. Since there is only one correct (winning) for a historical project, MRR can be prevalently used as the main

accuracy-based metric for team recommendation algorithms. The reciprocal rank of a query is defined mathematically as the inverse of the highest rank of relevant results retrieved. The mean reciprocal rank is the average of the reciprocal ranks of the query set Q's results. Formally, given testing set Q, let $rank_p$ be the rank of the first correct answer of the query $p \in Q$, then MRR of the query set Q is defined as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{23}$$

Here $rank_i$ refers to the rank of the first relevant result for the i-th query. Note that if the relevant result is not found, $\frac{1}{rank_i}$ is zero.

- **Mean Rank (MR)** is defined as the average rank of all the correctly predicted results. MR can be mathematically defined as:

$$MR = \frac{1}{|Q|} \sum_{p \in Q} rank_p \tag{24}$$

- **Mean Rank of Hits (Hit MR)** is defined as the average ranks of relevant results. Non-relevant results recommended by the system will not be taken into account. It can be mathematically defined as:

$$Hit\ MR = \frac{\sum_{p \in Q} \begin{cases} rank_q & \text{if } rank_q \text{ is found} \\ 0 & \text{otherwise} \end{cases}}{|\{p \in Q : rank_p \text{ is found}\}|} \tag{25}$$

- **Hit at 10** and **Hit at 100** are defined as the proportion of the number of relevant results found in the top 10 and 100 results, respectively. It indicates how likely the recommender would recommend the winning team within the top 10 and top 100 ranks, respectively.
- **Mean Average Precision (MAP)** is used to measure how precise the list of recommended answers is. In recommendation systems, the ranks of correct answers are crucial. Hence, the precision at rank K is computed at every top K of recommendation result to evaluate the precision of recommendation systems.

$$Precision@K = \frac{\sum_{i=1}^{K} H(t_0, t_i)}{K} \tag{26}$$

The reported precision@K is the average of the individual precision values from all the test samples in Q. Mean average precision (MAP) is the average of the precision@K, where K = 1, ..., 10.

$$MAP = \frac{\sum_{i=1}^{10} Precision@i}{10} \tag{27}$$

Table 3 shows the previous winning team retrieval of each algorithm in each dataset. WESTREC algorithm yields the highest MRR, Mean Rank, Hits at 10, and MAP in every dataset compared to other algorithms. From the results, it can be interpreted that WESTREC can recommend the teams that previously completed the corresponding projects in high rank. In terms of MRR values, WESTREC yields the best results. Moreover, WESTREC has the lowest (best) Hit MR and MR among the three approaches, implying that the relevant results, if found, are recommended in high rank. MAP measures how precisely the algorithm recommends relevant results, considering only the top 10 recommended teams. WESTREC yields the highest MAP, meaning that it can recommend the previous winning team precisely in the top ranks. However, when considering Hits at 100, WESTREC is worse than the other approaches. In contrast, WESTREC has the highest Hits at 10. This is because WESTREC seems confident in capturing certain teams, hence ranking them in the top results.

An example of a software team recommendation for a real software issue from Atlassian is shown in Table 4. The usernames are anonymized into UXXX code. The actual team of this project consists of Developer **U101** and Reviewer **U102**, with **U100** as the project assignee. The **bold** user code is the person who actually participated in the project. The teams with * are those that have successfully resolved at least one project before. This particular example illustrates that WESTREC not only can predict correct (whole and partial) teams in the top ranks but also can recommend teams with experience in successfully resolving previous projects.

RQ2: WESTREC has the best ranking ability for recommending the teams since it can recommend previous winning teams in better ranks in terms of MRR, Hit MR, Hits at 10, and MAP values.

4.3.3 Team effectiveness comparison by communication cost and teams characteristics analysis

To answer **RQ3**, the proposed metrics for team effectiveness (STEM) are calculated for the recommended teams by each approach and compared with those by WESTREC. Also, we separate the metric into five groups as described in Sect. 3.7. Then, we applied the One Way ANOVA test with $\alpha = 0.05$ between the values of metrics from each approach. Such statistical tests provide evidence on whether the distribution of the metrics in one approach is significantly different from the others. Since each group of the metrics represents a different aspect of a given software team, we discuss each group separately.

Communication Cost: Table 5 shows the average communication costs of the recommended teams by each approach and dataset. WESTREC has lower communication costs than Liu et al. (2014)'s method in every dataset. The P-values from Table 6 support that all of them are significantly different in distribution (p-value<0.05), except for the Cc-SD and Cc-LD of WESTREC and Recentness. Therefore, WESTREC outperforms Liu et al. (2014)'s approach in recommending teams with low communication costs (the team members are closer to each other). Apache and Atlassian contain 49 and 12 project domains, respectively. As

Table 3 Evaluation results of previous winning team retrieval using the recommendation evaluation protocol

	Apache				Atlassian			
	Recentness	Liu et al.	WESTREC	Improvement (%)	Recentness	Liu et al.	WESTREC	Improvement (%)
MRR	0.051	0.061	**0.117**	91.803	0.248	0.182	**0.616**	148.387
Hit MR	36.474	22.028	**1.307**	94.067	15.193	25.514	**2.840**	82.173
MR	**61.722**	90.657	88.255	− 42.988	**30.134**	46.328	30.677	− 1.802
Hits at 10	0.131	0.112	**0.189**	44.275	0.0806	0.478	**2.353**	392.259
Hits at 100	**0.937**	0.1914	0.190	− 79.723	**4.030**	3.249	3.213	− 20.273
MAP	0.046	0.057	**0.116**	103.509	0.157	0.101	**0.542**	245.223

Bold values represent the best evaluation value among the three methods (i.e., Recentness, Liu et al., WESTREC) corresponding to each evaluation metric

Table 4 Example of top 10 recommended teams by WESTREC and the baselines

Rank	Recentness	Liu et al.	WESTREC
1	U109, U109	U113, U114	***U101**, **U101**
2	U106, U109	U115, U114	***U101**, **U102**
3	U106, U106	U109, U114	**U101**, **U100**
4	U106, U110	U113, U116	U103, **U101**
5	U109, U106	U113, U117	**U101**, U103
6	U109, U110	U118, U114	**U101**, U104
7	U109, U111	U113, U119	U105, **U101**
8	U109, U112	U113, U108	U106, **U101**
9	U106, U111	***U101**, U114	**U101**, U107
10	U106, U112	U113, U109	U108, **U101**

a result, Recentness is likely to give lower communication cost values in a dataset with fewer project domains. This implies that in a smaller scale software development with fewer project domains, using an approach of recommending recent software practitioners who recently contributed to other projects is an effective approach if communication cost is a concern. However, when it comes to huge software systems containing many project domains, this method would not work as well as in smaller software systems. Hence, if the recommender is run separately on each project domain, it could help lower overall team communication costs since it can avoid introducing members from different domains. However, it could also affect the overall performance of the recommendation algorithm. This is because we observe that there are still software practitioners who work across project domains. Thus, training the recommendation model on each individual project separately could remove cases where a recommended team comprises members across different project domains. In conclusion, when considering the communication cost aspects, WESTREC outperforms the two approaches in large datasets (i.e., Apache and Atlassian).

Team Morale: Tables 7 and 8 show the average team morale metrics and the p-values of the Recentness and Liu et al. (2014)'s approaches compared to WESTREC. Similar to the communication costs comparison, WESTREC gives significantly higher team morale metrics than Liu et al. (2014)'s approach in every dataset and metric.

Team Cohesion: The average team cohesion metrics and the p-values derived from the comparisons of WESTREC against Recentness and Liu et al. (2014)'s approaches are shown in Tables 9 and 10. Similar to the team communication costs comparison and morale analyses, WESTREC performs better than Liu et al. (2014)'s approach in recommending teams with higher team cohesion in every dataset and team cohesion metric. As seen in Table 9, **Team Coherence**, **Relatedness** $_{u_A}$, and **Closeness** are higher than the rest of the metrics in team cohesion. This implies that all approaches give more importance to the team cohesion metrics in the historical collaboration than the project dependency. Therefore, the three approaches are likely to recommend teams that consist of software practitioners who have worked with their previous collaborators. On

Table 5 Average communication cost metrics of teams recommended by each approach

	Apache				Atlassian			
	Recentness	Liu et al.	WESTREC	Improvement (%)	Recentness	Liu et al.	WESTREC	Improvement (%)
Cc-R	7.732	5.932	**5.015**	15.459	2.186	2.612	**1.862**	14.822
Cc-Steiner	7.774	6.074	**5.100**	16.036	2.235	2.737	**1.966**	12.036
Cc-SD	8.140	6.597	**5.521**	16.310	2.405	3.148	**2.299**	4.407
Cc-LD	7.926	6.206	**5.193**	16.3229	2.278	2.843	**2.037**	10.580

Bold values represent the best evaluation value among the three methods (i.e., Recentness, Liu et al., WESTREC) corresponding to each evaluation metric

Table 6 *p* values from average communication cost metrics between WESTREC and the other baselines

	Apache		Atlassian	
	Recentness	Liu et al.	Recentness	Liu et al.
Cc-R	< 0.001	< 0.001	0.006	< 0.001
Cc-Steiner	< 0.001	< 0.001	0.035	< 0.001
Cc-SD	< 0.001	< 0.001	0.611	< 0.001
Cc-LD	< 0.001	< 0.001	0.088	< 0.001

the other hand, the three approaches are less likely to recommend teams based on the project's dependencies.

Team Experience: Tables 11 and 12 show the average team experience metrics, and the p-values of the Recentness and Liu et al. (2014)'s approaches compared to WESTREC. It is evident that teams recommended by WESTREC have significantly higher team experience compared to those of the other two baselines.

Team Proficiency: Table 13 shows the average team proficiency metrics of the recommended teams by each approach and for each dataset. Table 14 shows the p-values from performing the statistical test between WESTREC and the two baselines. The p-values of the skill competency in the three approaches in every dataset are higher than 0.05. Therefore, we could not conclude that there are significant differences in the skill competency of the teams recommended by the three approaches. Regardless, Recentness gives higher skill diversity in the Apache and Atlassian datasets. Also, Liu et al. (2014)'s method gives higher skill diversity than WESTREC in every dataset. To explain such a phenomenon, we found the same trend in the communication cost, team coordination, and team cohesion comparison: Recentness performs best in a small software system and worst in a larger software system.

From these findings, WESTREC gives higher values of metrics than Liu et al. (2014)'s approach in three out of five groups of metrics. Furthermore, we can observe that none of the approaches gives high values of team effectiveness in every aspect. It implies that different approaches focus on the different characteristics when recommending a team. However, when considering a large-scale software system, WESTREC outperforms the other two approaches (i.e., WESTREC gives the highest values of team effectiveness metrics in three of five groups associated with the different characteristics of team effectiveness).

RQ3: From our experiments, WESTREC outperforms the other approaches (i.e., Recentness and Liu et al. (2014)'s approaches) in three of five groups of metrics associated with different characteristics of teams in larger software systems.

Table 7 Average values of team morale metrics of teams recommended by each approach

	Apache				Atlassian			
	Recentness	Liu et al.	WESTREC	Improvement (%)	Recentness	Liu et al.	WESTREC	Improvement (%)
Team Positivity	0.007	0.008	**0.016**	106.809	0.006	0.001	**0.007**	13.071
Team Criticism	0.006	0.007	**0.14**	105.566	0.007	0.001	**0.008**	15.901
Team Positivity$_{u_A}$	0.006	0.008	**0.016**	112.989	0.002	0.002	**0.008**	292.262
Team Criticism$_{u_A}$	0.006	0.007	**0.015**	111.586	0.004	0.002	**0.011**	184.262
Connection	0.001	0.001	**0.002**	63.560	0.004	0.002	**0.008**	109.474

Bold values represent the best evaluation value among the three methods (i.e., Recentness, Liu et al., WESTREC) corresponding to each evaluation metric

Table 8 p values from the average team morale metrics between WESTREC and the other baselines

	Apache		Atlassian	
	Recentness	Liu et al.	Recentness	Liu et al.
Team Positivity	< 0.001	< 0.001	0.696	< 0.001
Team Criticism	< 0.001	< 0.001	0.643	< 0.001
Team Positivity$_{u_A}$	< 0.001	< 0.001	< 0.001	< 0.001
Team Criticism$_{u_A}$	< 0.001	< 0.001	0.005	< 0.001
Connection	< 0.001	< 0.001	0.010	< 0.001

5 Analysis of the actual teams

This section discusses the result of analyzing the actual winning teams using STEM. Since there are many metrics in each dimension of STEM (i.e., CC, TM, TC, TE, and TP), the representative metrics for each dimension are introduced to facilitate the analysis of team characteristics.

Furthermore, we introduce Total Team Effectiveness (TTE) to represent the overall effectiveness of a team, calculated using the following formula.

$$TTE = \frac{InvCC + TM + TC + TE + TP}{5} \tag{28}$$

$InvCC = \frac{1}{CC}$ is the inversion of the representative communication cost. We use inversion because a lower communication cost indicates better team effectiveness. Note that the representation metrics, except communication cost, undergo Min-Max normalization to keep the value between [0, 1]. Table 15 shows the mean and median of the STEM statistics calculated from the actual winning (Win) and not winning (NotWin) teams from the Apache and Atlassian datasets. The p-values from the Mann-Whitney U tests between the Win and NotWin populations are also reported using $\alpha = 0.05$.

It is to be anticipated that the TM, TC, TE, TP, and TTE statistics will be greater for the Win teams as compared to the NotWin teams. Additionally, the CC will be lower for the Win teams. For Apache, it is evident that the means and medians of all the statistics are favorable for the Win teams, with CC 42.60% lower, TM 25.73%, TC 37.32%, TE 78.46%, TP 27.65%, and TTE 33.13% higher than the NotWin population on average. In addition, the statistical tests demonstrate that there is a statistically significant difference in the statistics that were produced using data from the two groups. The most significant difference can be seen between the Win and NotWin teams in terms of TE (Team Experience), but TM (Team Morale) is the factor that both sets of teams have in common and produces the lowest gap.

Although not as prominent as Apache's, the STEM statistics may still be considered reflective of Atlassian's teams. The statistical tests indicate that the only characteristics with statistically significant differences are CC, TC, and TTE. On average, the Win teams have a CC score 16.66% lower than the NotWin teams. The average TM, TE, TP, and TTE of the Win teams are greater than those of the NotWin teams by 41.08%, 0.19%, 17.14%, and 19.29%, respectively. It is noteworthy to observe

Table 9 Average values of team cohesion metrics of teams recommended by each approach

	Apache				Atlassian			
	Recentness	Liu et al.	WESTREC	Improvement (%)	Recentness	Liu et al.	WESTREC	Improvement (%)
Co-Project Frequency	0.000	**0.001**	**0.001**	38.334	0.009	0.009	**0.031**	257.899
Project Proximity	0.001	0.003	**0.005**	79.439	0.024	0.015	**0.053**	118.426
Team Coherence	0.021	0.057	**0.078**	36.945	0.159	0.119	**0.204**	27.841
Relatedness_{u_A}	0.020	0.052	**0.071**	37.143	0.146	0.111	**0.180**	23.359
Closeness	0.115	0.139	**0.180**	28.841	0.421	0.363	**0.549**	30.312

Bold values represent the best evaluation value among the three methods (i.e., Recentness, Liu et al., WESTREC) corresponding to each evaluation metric

Table 10 p values from the average team cohesion metrics between WESTREC and the other baselines

	Apache		Atlassian	
	Recentness	Liu et al.	Recentness	Liu et al.
Co-Project Frequency	< 0.001	< 0.001	< 0.001	< 0.001
Project Proximity	< 0.001	< 0.001	< 0.001	< 0.001
Team Coherence	< 0.001	< 0.001	0.035	< 0.001
Relatedness$_{u_A}$	< 0.001	< 0.001	0.066	< 0.001
Closeness	< 0.001	< 0.001	< 0.001	< 0.001

that the average TM (Team Morale) of the NotWin teams is 0. This suggests that these teams may not have interacted with each other at all, either positively or critically, in their previous tasks. The difference between the Win teams and the NotWin teams in terms of TC (Team Cohesion) is the largest on average; the Win teams have a score that is 41.08% higher than the NotWin group, which suggests that the successful teams have worked together frequently in the past. The difference in TE (Team Experience), which is the lowest of the three factors, is just 0.19%, which suggests that the members of the team's cumulative winning experience may not add to the efficiency of the collaboration.

It is noteworthy to observe that the scores for TM (Team Morale) are relatively low in both the Win and NotWin teams of Apache and Atlassian. This might be due to the fact that such a statistic is generated based on the Team Interaction (Eq. 8) formula, in which the denominator (i.e., $|T| \times (|T| - 1)$) could be much bigger than the nominator. In any case, despite the fact that TM may not be a reliable measurement in an absolute sense, it is still feasible to use such a statistic for relative comparison.

In practice, we would recommend utilizing TTE as a means of measuring the overall performance of teams since it has been demonstrated to be a clear discriminator between Win and NotWin teams, with relative differences of 33.13% and 19.29% on average for Apache and Atlassian, respectively. Additionally, the results of the statistical tests demonstrate that the Win and NotWin groups in each dataset have TTE values that are statistically different from one another.

We also perform further analysis on the top teams that achieve the best TTE in each software system. Table 16 shows the teams that have the highest TTE in each software system. From our observation, all of the top five teams from five different projects are the same team consisting of two members working in multiple roles (e.g., userA works as the project assignee and developer while userB works as the reviewer and tester). As they work together multiple times, the metrics that capture communication costs, team morale, and cohesion are very high. Besides, we observed that they also participated in and successfully resolved projects together (i.e., most of the features, such as team win-rate and experience, are very high). Thus, this is a good anecdotal example of a successful team. The difference in TTE of each project in the same team is caused by the project familiarity, component experience, and role experience. Since there are different projects, the content, components, and roles may vary. In Apache, the teams in the top five projects are

Table 11 Average values of team experience metrics of teams recommended by each approach

	Apache				Atlassian			
	Recentness	Liu et al.	WESTREC	Improvement (%)	Recentness	Liu et al.	WESTREC	Improvement (%)
Project Familiarity	**0.483**	0.443	0.446	− 7.789	**0.483**	0.443	0.446	0.349
Component Experience	0.046	0.046	**0.047**	0.188	0.042	0.039	**0.045**	5.395
Project Domain Experience	0.104	**0.105**	**0.105**	− 0.356	0.104	**0.105**	**0.105**	− 2.710
Team Contribution	**0.362**	0.257	0.258	− 28.753	0.250	0.234	**0.256**	2.641
Experience	**0.136**	0.132	0.126	− 7.279	0.084	0.073	**0.104**	24.630
Win Rate	0.874	**0.881**	0.871	− 1.103	**0.943**	0.909	0.940	− 0.293
Role Experience	**0.162**	0.159	0.153	− 5.683	0.139	0.131	**0.161**	15.544

Bold values represent the best evaluation value among the three methods (i.e., Recentness, Liu et al., WESTREC) corresponding to each evaluation metric

Table 12 p values from the average team experience metrics between WESTREC and the other baselines

	Apache		Atlassian	
	Recentness	Liu et al.	Recentness	Liu et al.
Project Familiarity	0.286	< 0.001	0.893	< 0.001
Component Experience	0.662	0.951	0.810	0.521
Project Domain Experience	0.682	0.859	0.853	0.489
Team Contribution	< 0.001	0.867	0.692	0.203
Experience	< 0.001	0.004	0.012	< 0.001
Win Rate	0.055	< 0.001	0.692	0.00
Role Experience	< 0.001	0.016	0.089	0.020

different. As we can see from Table 16, the assignees in top projects pick teams that perform well in TM, TC, and TP. However, the TE is quite low compared to other dimensions of team effectiveness in the top projects. This shows that the teams formed may not have extensive relevant experience but are proficient in the project domain and collaborate well. This also occurs in Atlassian; multiple project assignees formulate teams with low team experience. However, some of the top teams in Atlassian have 0 TM. This indicates that they are not communicating with each other in Jira.

6 Threats to validity

We discuss the threats to the validity of our research as follows:

- **Team member roles** are only collected as they are explicitly labeled in a project. However, there could be a person who participated in a project without an explicit role in Jira. This could lead to a threat to internal validity. For example, some software systems, such as Apache, do not specify Developer roles in Jira. Thus, we mitigate this by considering users who commit to the project's GitHub repository shown in the Jira as developers.
- **Recommended teams** may differ each time Max-Logit is executed. This is because Max-Logit introduces randomness in the candidate selection part of the algorithm. This could be solved by increasing the number of iterations in Max-Logit; however, this could result in a much longer computational time. We thus mitigate this threat by setting the same random seed in all of our experiments.
- **Measurement of team effectiveness** could be conducted differently from ours. To the best of our knowledge, there does not exist a formal way to calculate the metrics for software team effectiveness. For example, several different communication cost formulas focus on different aspects of communication-based team performance. However, such metrics do not fully reflect other aspects of a software team, such as collective skills and team cohesion. Therefore, the team

Table 13 Average values of the team proficiency metrics of teams recommended by each approach

	Apache				Atlassian			
	Recentness	Liu et al.	WESTREC	Improvement (%)	Recentness	Liu et al.	WESTREC	Improvement (%)
Skill Diversity	**0.927**	0.883	0.865	− 6.673	**0.809**	0.785	0.740	− 8.553
Skill Competency	0.384	0.382	**0.388**	1.005	**0.167**	0.160	0.151	− 9.531

Bold values represent the best evaluation value among the three methods (i.e., Recentness, Liu et al., WESTREC) corresponding to each evaluation metric

Table 14 *p* values from the average team proficiency metrics between WESTREC and the other baselines

	Apache		Atlassian	
	Recentness	Liu et al.	Recentness	Liu et al.
Skill Diversity	< 0.001	< 0.001	< 0.001	< 0.001
Skill Competency	0.345	0.147	0.589	0.770

Table 15 Statistics of STEM calculated from actual winning (Win) and not winning (NotWin) teams

		Mean			Median			
		NotWin	Win	Δ%	NotWin	Win	Δ%	*p*-value
Apache	CC	1.7422	1.0	− 42.60	1.0	1.0	0	< 0.05
	TM	0.0447	0.0562	25.73	0.0000	0.0094	–	< 0.05
	TC	0.3435	0.4717	37.32	0.3569	0.4768	33.59	< 0.05
	TE	0.2887	0.5152	78.46	0.2678	0.4918	83.64	< 0.05
	TP	0.5118	0.6533	27.65	0.5642	0.6903	22.35	< 0.05
	TTE	0.4051	0.5393	33.13	0.4161	0.5318	27.81	< 0.05
Atlassian	CC	1.2	1.0	− 16.66	1.0	1.0	0	< 0.05
	TM	0.0000	0.0776	–	0.0000	0.0000	0	> 0.05
	TC	0.2958	0.4173	41.08	0.3592	0.4816	34.08	< 0.05
	TE	0.3097	0.3103	0.19	0.3013	0.2991	− 0.73	> 0.05
	TP	0.4311	0.505	17.14	0.4247	0.4792	12.83	> 0.05
	TTE	0.3873	0.462	19.29	0.4164	0.4483	7.66	< 0.05

Table 16 Top 5 projects with teams that have highest TTE in each software system

	inverseCC	TM	TC	TE	TP	TTE
Apache						
A	1.000	1.000	0.739	0.418	0.770	0.785
B	1.000	0.779	0.743	0.495	0.681	0.739
C	1.000	0.487	0.741	0.788	0.648	0.733
D	1.000	0.586	0.745	0.567	0.691	0.718
E	1.000	0.652	0.742	0.423	0.734	0.710
Atlassian						
A	1.000	1.000	0.881	0.318	0.860	0.812
B	1.000	1.000	0.881	0.313	0.439	0.727
C	1.000	0.000	0.876	0.421	0.868	0.633
D	1.000	0.556	0.881	0.285	0.439	0.632
E	1.000	0.000	0.878	0.234	0.941	0.611

effectiveness metrics can be studied further to establish formal formulas for the metrics.

- **The teams in our datasets** may not be optimal teams for working on the projects. Hence, this can affect our team scoring model since it learns from the actual team who worked on the projects. This could limit the capability of WESTREC since finding the optimal teams for working on a project is difficult, especially where the ground-truth information that indicates ideal team configurations for specific tasks is difficult to acquire or may not even exist.
- **The open-source project setting** would not be representative of all kinds of software projects, especially in a commercial setting where developers commit to working under legitimate contracts, and inherently result in less failure rate than the open-source settings where most of the efforts are voluntary. In light of this, more research into the performance of our suggested method for commercial projects is needed before it can be generalized to alternative paradigms of collaborative software development.

7 Related work

Our research aims to develop the recommendation algorithm for software teams and evaluate their effectiveness. In this section, we first discuss the related work on single-role recommendations. Then, we discuss the related literature on team recommendation systems. Lastly, we discuss the related literature on measuring the effectiveness of teams in different domains.

7.1 Single-role recommendations

Automatic recommendation of software experts for a particular function (e.g., developer, reviewer, tester) has been pivotal to the software engineering communities. However, since our work primarily concerns quantifying the effectiveness of recommended software teams rather than individuals, we briefly discuss relevant work on the single-role recommendation.

Rahman et al. (2016) proposed a code reviewer recommendation tool, namely CORRECT. The authors highlighted the idea that the developers who experienced reviewing similar or relevant past pull requests are suitable candidates for reviewing upcoming pull requests. The tools consider the relevant cross-project work experience of a developer (e.g., external libraries used) and the developer's experience in certain specialized technologies (e.g., search, Google Cloud, pipeline, etc.) that are associated with a pull request as features for recommending reviewers to a pull request on GitHub repositories. They used cosine-similarity to calculate the similarity between the pull request and recommend reviewers based on their experience with similar pull requests. Although this work only considered the experience of a reviewer, the idea of recommending a person who has experience in similar tasks can be applied to our research when calculating features for the team scoring function.

Ye et al. (2018) proposed a teammate recommendation algorithm for crowdsourced software development. They found that developers benefit greatly from

team collaboration, and it is common to see developers fail in teammate-seeking. They considered closeness with teammates, their expertise, and expertise gained through collaboration as a factor for recommending teammates. They designed an approximation algorithm to find suitable teammates for a developer. We adopted the idea of team collaboration and expertise as one of the features for calculating team scores. Compared to our research, this work does not consider the role of a team.

Rebai et al. (2020) proposed reviewer recommendation algorithm. They consider the expertise of reviewers in file level, availability of reviewers, and history of collaboration files between reviewers and developers. They formulated the problem as a multi-objective search problem and applied multiple variants of the multi-objective search algorithm to recommend a list of reviewers. Although authors only focus on recommending reviewers for a software task, we can adopt the idea of expertise and history of collaboration to our research.

Zhang et al. (Jan 2020) proposed a developer recommendation system for the Topcoder challenge, which is a crowd-sourcing software development platform. They adopt a meta-learning-based policy model (e.g., ExtraTrees, XGBoost, and Neural Network) for recommending developers. They consider the Topcoder challenge (e.g., challenge duration) and developer features (i.e., technical skill and historical performance) as features. The idea of considering both contexts of the task and developer skill and historical performance can be applied to our research. However, this work does not mention the collaboration of a team.

7.2 Team recommendations

Hupa et al. (2010) introduced a method to compute a score for a team based on the members' social context and their suitability for a particular project. They proposed a three-dimensional social network that was used to represent the social context of individuals. The network consisted of a knowledge network, a trust network, and an acquaintance network. A knowledge network is a two-mode graph with an edge connecting people to certain skills. The weight of an edge represents the level of the corresponding skills a person has. The trust network is a directed graph with an edge connecting a person with another. The weight of an edge represents how much a person trusts another person. The acquaintance network is an undirected graph with an edge connecting one person to another. The weight of an edge represents the intensity of the interactions between two people. The graphs are used to quantify the following features: Closeness Centrality, Average Interpersonal Trust Measure, Aggregated Skill Difference, and Maximum Skill Distribution. After defining the criteria to score a team, they combined all criteria into one function and optimized it by using the reference point method. The idea of the multidimensional graph can be applied to the software team's recommendation problem; however, this work does not provide a method to find the best team from a combinatorial solution space.

Datta et al. (2011a) developed an expert recommendation system. This system allows users to input a set of required skills and form teams based on such skill requirements. In addition, the user could also filter people out by adding several filtering criteria, such as school, status, etc. In addition, users could rank the

importance of each skill through the configurable recommendation parameters, such as the number of people in a team, their cohesiveness, importance, etc. In order to determine the quality of the team, the system considered two features: competence coverage which focuses on the team's skills as a whole, and team cohesiveness which is derived from the team social graph. The idea of customizable parameters such as the importance of a skill can be useful for flexible and personalized recommendations; however, this work does not mention any team formulation algorithms, which are important for large datasets.

Liu et al. (2014) proposed an approach to recommend a team with given roles. They considered both individual strengths and team strengths as features. They, therefore, developed a team strength function that is used to determine both the compatibility and strength of a given team. They divided features into two types, namely individual strengths and team features. Individual strength features are features that describe a person's strength without considering the team. The individual strength features consist of experience, win experience, win rate, and roles. The team features are the features that describe the compatibility of a team. The team features consist of team closeness and social connections. The weights of the features were optimized by logistic regression trained on historical project outcomes. Because it was infeasible to compute all the combinations of teams, they used a Max-Logit algorithm, a statistical learning-based algorithm studied in the potential game theory, to find the approximate best team. Their findings supported the expandability of the team formation algorithm. As our research has a similar problem setting, the idea of using the Max-Logit algorithm to enumerate possible candidate teams while using the features of a team as optimizing criteria is potentially applicable to our problem.

Alberola et al. (2016) presented an artificial intelligence tool for heterogeneous team formation in classrooms. They formed teams using Belbin's role theory and artificial intelligence techniques to handle the uncertainty of the student role and solve huge computation problems. They also considered the information that was collected before and after working on the teams. However, applying this solution may not be suitable for the software development process since it would introduce an additional burden to the developers, as they need to post-evaluate teams after task completion to check whether the task has been successfully executed.

Qiao et al. (2021) examined the team formation behaviors of physicians in online health communities (OHCs). They found that group joining behavior positively influences each physician's online demand and reputation. They also found that the higher level of person-group fit indicates better individual online demand and reputation in online health communities; however, physicians' title negatively impacts the relationship between team formation and the online demand. These findings can be potentially used to recommend teams of physicians appropriate to a given online community.

Recent research by Tuarob et al. (2021) has proposed that a software team may be characterized by a collection of characteristics that are derived from collaborative software graphs. Then, in the circumstances involving collaborative software development, these capabilities were put to use in the development of a recommendation algorithm for both particular roles and whole software teams. Their work differs from ours in two significant ways. First, the definition of a good team, according to

their study, is a team that has completed the task, while we enhanced the definition to narrow it down to teams that not only have completed the tasks but also did not result in any delays or task reopenings. Moreover, the primary objective of our research is to develop measures that might be utilized to automatically assess the effectiveness of recommended teams in various aspects. Finally, we demonstrated that the five proposed aspects of measuring team effectiveness are discriminative that can distinguish the desirable teams from the ineffective ones.

7.3 Team effectiveness

Beaver and Schiavone (2006) studied the influence of skill and experience on the final quality of software products. Their study was performed on the data from 26 small-scale software development projects. They have found that the development team skill is a significant factor in the adequacy of design and implementation. Furthermore, their results imply that the assignment of inexperienced software developers to tasks ill-suited to their skills significantly adversely affects the quality of the software product. Therefore, we adopt their findings that skills and experience are important to the quality of software products. This can imply that they also affect the effectiveness of the team. Hence, measurements of experience and skills are used as one of the metrics in our research.

Wang et al. (2015) empirically compared and contrasted many state-of-the-art team formation algorithms in the social network. They proposed the Unified System for Team Formation (USTF) and evaluated its performance on many datasets. They also studied the proposed metrics for team formation algorithms, including the communication cost with different formulas (i.e., CC-R, CC-Steiner, CC-SD, and CC-LD). Although their system and the state-of-the-art team formation algorithms aim to recommend teams that cover a specific set of skills considering team collaboration and communication, none of them consider the role requirements as criteria to form a team. Furthermore, such team formation algorithms do not fix the recommended team size. Therefore, we could not apply their team formation algorithm for our research since WESTREC aims to recommend teams with a given number of team members and roles. Instead, we adopt their idea of using the communication cost to compare the team formation algorithm in our metrics.

Dingsøyr et al. (2016) studied the factors which affect team performance. They reviewed literature in many research disciplines (i.e., management science, organizational psychology, and information systems) that study team effectiveness and proposed five factors that strongly affect team performance: team coordination, goal orientation, team cohesion, shared mental models, and team learning. They also compared their propositions to Agile Manifesto's 12 principles. Their findings illuminate the path forward to improve the effectiveness and productivity of the team. We concur with their results that coordination and cohesiveness within the team are among the most critical aspects that influence team performance. Our proposed metrics for measuring team morale and cohesion also embrace similar concepts in their research.

Recently, Yang et al. (2021) studied the impact of team diversity on the team performance of physicians in online health communities. A set of fixed-effect models was used to measure the team diversity's effects on team performance from the signal transmission. They found that the diversity of reputation and experience has positive impacts on team performance. However, the presence of star physicians has different effects on the relationship between team diversity and team performance. Despite the fact that their research did not specifically address software teams, we agree with their conclusion that diversity is likely one of the most important factors in productive cooperation. Both the team experience and the team proficiency metrics that we propose in this study have a similar understanding of the importance of having a diverse and experienced team.

8 Conclusion and future work

We proposed WESTREC, a machine learning based algorithm to automatically recommend software teams given a project assignee and project description. Empirical evaluations of the two well-known open-source software systems show that WESTREC outperforms the baseline approaches (i.e., Recentness and Liu et al. (2014)'s approaches). Our results support that WESTREC performs best when applying to large-scale software development since three out of five groups of software team effectiveness metrics (STEM), which characterize different aspects of software teams, are significantly better than those of the baselines. Furthermore, using Random Forest to model the Team Effectiveness scoring function allows WESTREC to significantly improve the team ranking ability, compared to Liu et al. (2014)'s approach. WESTREC performs well in recommending previously successful teams in solving projects (i.e., winning teams) in top ranks. In future directions, we aim to investigate incorporating deep learning techniques in software team recommendations. Furthermore, we plan to apply the proposed team effectiveness metrics to diverse software development environments such as software projects in educational organizations and software companies to gauge their ability to generalize to different software collaboration cultures.

Acknowledgements This research is supported by the Thailand Science Research and Innovation (TSRI), formerly known as Thailand Research Fund (TRF), and the National Research Council of Thailand (NRCT) through Grant No. RSA6280105. We also appreciate computing resources from Mahidol University (Grant No. MU-MiniRC02/2564).

References

Akbar, M.A., Sang, J., Khan, A.A., Mahmood, S., Qadri, S.F., Hu, H., Xiang, H.: Success factors influencing requirements change management process in global software development. J. Comput. Lang. **51**, 112–130 (2019)

Alberola, J.M., Del Val, E., Sanchez-Anguix, V., Palomares, A., Teruel, M.D.: An artificial intelligence tool for heterogeneous team formation in the classroom. Knowl.-Based Syst. (2016). https://doi.org/10.1016/j.knosys.2016.02.010

Alsharo, M., Gregg, D., Ramirez, R.: Virtual team effectiveness: the role of knowledge sharing and trust. Inf. Manag. **54**, 11 (2016). https://doi.org/10.1016/j.im.2016.10.005

Assavakamhaenghan, N., Choetkiertikul, M., Tuarob, S., Kula, R.G., Hata, H., Ragkhitwetsagul, C., Sunetnanta, T., Matsumoto K.: Software team member configurations: a study of team effectiveness in moodle. In: Proceedings of the 10th International Workshop on Empirical Software Engineering in Practice (IWESEP), pp. 19–195 (2019). https://doi.org/10.1109/IWESEP49350.2019.00012

Beaver, J., Schiavone, G.: The effects of development team skill on software product quality. ACM SIGSOFT Softw. Eng. Notes **31**, 1–5 (2006). https://doi.org/10.1145/1127878.1127882

Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**(Jan), 993–1022 (2003)

Chen, R., Liang, C., Gu, D., Leung, J.Y.: A multi-objective model for multi-project scheduling and multi-skilled staff assignment for it product development considering competency evolution. Int. J. Prod. Res. **55**(21), 6207–6234 (2017)

Chipulu, M., Ojiako, U., Gardiner, P., Williams, T., Mota, C., Maguire, S., Shou, Y., Stamati, T., Marshall, A.: Exploring the impact of cultural values on project performance—the effects of cultural values, age and gender on the perceived importance of project success/failure factors. Int. J. Oper. Prod. Manag. **34**, 364–389 (2014). https://doi.org/10.1108/IJOPM-04-2012-0156

Choetkiertikul, M., Dam, H.K., Tran, T., Ghose, A.: Predicting the delay of issues with due dates in software projects. Empir. Softw. Eng. **22**(3), 1223–1263 (2017). https://doi.org/10.1007/s10664-016-9496-7

Chow, T., Cao, D.-B.: A survey study of critical success factors in agile software projects. J. Syst. Softw. **81**(6), 961–971 (2008). https://doi.org/10.1016/j.jss.2007.08.020

Colazo, J.: Collaboration structure and performance in new software development: findings from the study of open source projects. Int. J. Innov. Manag. **14**, 735–758 (2010). https://doi.org/10.1142/S1363919610002866

Datta, A., Tan Teck Yong, J., Ventresque, A.: T-recs: team recommendation system through expertise and cohesiveness. In: Proceedings of the 20th International Conference Companion on World Wide Web, WWW '11, New York, NY, USA, pp. 201–204. ACM. ISBN 978-1-4503-0637-9 (2011a). https://doi.org/10.1145/1963192.1963289

Datta, A., Yong, J., Ventresque, A.: T-recs: team recommendation system through expertise and cohesiveness, pp. 201–204 (2011b). https://doi.org/10.1145/1963192.1963289

Dingsøyr, T., Dybå, T.: Team effectiveness in software development: human and cooperative aspects in team effectiveness models and priorities for future studies (2012). https://doi.org/10.1109/CHASE.2012.6223016

Dingsøyr, T., Fægri, T., Dybå, T., Haugset, B., Lindsjørn, Y.: Team performance in software development: research results versus agile principles. IEEE Softw. **33**, 106–110 (2016). https://doi.org/10.1109/MS.2016.100

Dubinsky, Y., Hazzan, O.: Roles in agile software development teams, pp. 157–165 (2004). https://doi.org/10.1007/978-3-540-24853-8_18

Fagerholm, F., Ikonen, M., Kettunen, P., Münch, J., Roto, V., Abrahamsson, P.: How do software developers experience team performance in Lean and Agile environments? In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10. ISBN 9781450324762 (2014). https://doi.org/10.1145/2601248.2601285

Faraj, S., Sproull, L.: Coordinating expertise in software development teams. Manag. Sci. **46**, 1554–1568 (2000). https://doi.org/10.1287/mnsc.46.12.1554.12072

Foster, E.C.: Human resource management. In: Software Engineering, pp. 253–269. Springer (2014)

Franzago, M., Di Ruscio, D., Malavolta, I., Muccini, H.: Collaborative model-driven software engineering: a classification framework and a research map. IEEE Trans. Softw. Eng. **1–1**, 09 (2017). https://doi.org/10.1109/TSE.2017.2755039

Grigore, M., Rosenkranz, C.: Increasing the willingness to collaborate online: an analysis of sentiment-driven interactions in peer content production. In: Galletta, D.F., Liang, T. (eds.) Proceedings of the International Conference on Information Systems, ICIS 2011, Shanghai, China, December 4–7, 2011. Association for Information Systems (2011)

Guha, R., Kumar, R., Raghavan, P., Tomkins, A.: Propagation of trust and distrust. In: Proceedings of the 13th International Conference on World Wide Web, pp. 403–412 (2004)

Günsel, A., Açikgšz, A., Tükel, A., Öğüt, E.T.: The role of flexibility on software development performance: an empirical study on software development teams. Procedia Soc. Behav. Sci. **58**, 853–860 (2012). https://doi.org/10.1016/j.sbspro.2012.09.1063

Han, W.-M., Huang, S.-J.: An empirical analysis of risk components and performance on software projects. J. Syst. Softw. **80**(1), 42–50 (2007). https://doi.org/10.1016/j.jss.2006.04.030

Huckman, R., Staats, B., Upton, D.: Team familiarity, role experience, and performance: evidence from Indian software services. Manag. Sci. **55**, 85–100 (2009). https://doi.org/10.1109/EMR.2012.6172773

Hupa, A., Rzadca, K., Wierzbicki, A., Datta, A.: Interdisciplinary matchmaking: choosing collaborators by skill, acquaintance and trust, pp. 319–347 (2010)

iDalko: A guide to Jira workflow best practices (2018). https://www.idalko.com/jira-workflow-best-practices/

Jiang, J., Klein, G.: Software development risks to project effectiveness. J. Syst. Softw. **52**, 3–10 (2000). https://doi.org/10.1016/S0164-1212(99)00128-4

Jiang, J.J., Klein, G., Means, T.L.: Project risk impact on software development team performance. Proj. Manag. J. **31**(4), 19–26 (2000). https://doi.org/10.1177/875697280003100404

Kale, A.: Modeling trust and influence in blogosphere using link polarity. Master's thesis, April (2007)

Khan, A.A., Basri, S., Dominc, P.: A proposed framework for communication risks during RCM in GSD. Procedia—Social and Behavioral Sciences **129**, 496–503 (2014). In: 2nd International Conference on Innovation, Management and Technology Research

Khan, A.A., Keung, J., Hussain, S., Niazi, M., Tamimy, M.M.I.: Understanding software process improvement in global software development: a theoretical framework of human factors. SIGAPP Appl. Comput. Rev. **17**(2), 5–15 (2017)

Khan, A.A., Keung, J., Niazi, M., Hussain, S., Ahmad, A.: Systematic literature review and empirical investigation of barriers to process improvement in global software development: client–vendor perspective. Inf. Softw. Technol. **87**, 180–205 (2017)

Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks, pp. 467–476 (2009). https://doi.org/10.1145/1557019.1557074

Lindsjørn, Y., Sjøberg, D.I., Dingsøyr, T., Bergersen, G.R., Dybå, T.: Teamwork quality and project success in software development: a survey of agile development teams. J. Syst. Softw. **122**, 274–286 (2016). https://doi.org/10.1016/j.jss.2016.09.028

Liu, H., Qiao, M., Greenia, D., Akkiraju, R., Dill, S., Nakamura, T., Song, Y., Motahari Nezhad, H.R.: A machine learning approach to combining individual strength and team features for team recommendation (2014). https://doi.org/10.13140/2.1.4558.4966

Maalej, W., Ellmann, M., Robbes, R.: Using contexts similarity to predict relationships between tasks. J. Syst. Softw. **128**, 267–284 (2017). https://doi.org/10.1016/j.jss.2016.11.033

Monderer, D., Shapley, L.: Potential games. Games Econ. Behav. **14**, 124–143 (1996). https://doi.org/10.1006/game.1996.0044

Mudrack, P.: Defining group cohesiveness: a legacy of confusion? Small Group Res **20**, 37–49 (1989). https://doi.org/10.1177/104649648902000103

Naguib, H., Narayan, N., Brugge, B., Helal, D..: Bug report assignee recommendation using activity profiles. In: Proceeding of the 10th Working Conference on Mining Software Repositories (MSR), pp. 22–30. IEEE, May 2013. ISBN 978-1-4673-2936-1. http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6623999 (2013). https://doi.org/10.1109/MSR.2013.6623999

Niazi, M., Babar, M.A., Verner, J.M.: Software process improvement barriers: A cross-cultural comparison. *Inf. Softw. Technol.*, **52** (11):1204–1216 (2010). Special Section on Best Papers PROMISE 2009

Niazi, M., Mahmood, S., Alshayeb, M., Qureshi, A.M., Faisal, K., Cerpa, N.: Toward successful project management in global software development. Int. J. Proj. Manag. **34**(8), 1553–1567 (2016)

Oliver Bossert,J. L., Kretzberg, Alena.: Agile compendium, chapter 1.3, p 30. McKinsey Quarterly, 10 (2018)

Qiao, W., Yan, Z., Wang, X.: Join or not: The impact of physicians' group joining behavior on their online demand and reputation in online health communities. Inf. Process. Manag. **58**(5), 102634 (2021)

Rahman, M.M., Roy, C.K., Redl, J., Collins, J.A.: Correct: code reviewer recommendation at github for vendasta technologies. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, New York, NY, USA, pp. 792–797. Association for Computing Machinery. ISBN 9781450338455 (2016). https://doi.org/10.1145/2970276.2970283. URL

Rebai, S., Amich, A., Molaei, S., Kessentini, M., Kazman, R.: Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. Autom. Softw. Eng. **27**(3), 301–328 (2020). https://doi.org/10.1007/s10515-020-00275-6

Sokolov, E.: On software development product management: feature selection and model analysis for predicting Jira issue attributes (2017)

Sommerville, I.: Software Engineering, 9th edn. Pearson Education, London (2011)

Song, Y., Wong, S., Lee, K.-W.: Optimal gateway selection in multi-domain wireless networks: a potential game perspective, pp. 325–336 (2011). https://doi.org/10.1145/2030613.2030650

Storey, M., Zagalsky, A., Filho, F.F., Singer, L., German, D.M.: How social and communication channels shape and challenge a participatory culture in software development. IEEE Transa. Softw. Eng. **43**(2), 185–204 (2017)

Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**(4), 341–359 (1997)

The Standish Group: Chaos report 2015. Technical report, The Standish Group International, Inc (2015)

Tian, Y. Wijedasa, D., Lo, D., Le Gouesy, C.: Learning to rank for bug report assignee recommendation. In: Proceedings of the 24th International Conference on Program Comprehension (ICPC), pp. 1–10. ISBN 9781509014286 (2016). https://doi.org/10.1109/ICPC.2016.7503715

Tuarob, S., Assavakamhaenghan, N., Tanaphantaruk, W., Suwanworaboon, P., Hassan, S.-U., Choetkiertikul, M.: Automatic team recommendation for collaborative software development. Empir. Softw. Eng. **26**(4), 1–53 (2021)

Wang, X., Zhao, Z., Ng, W.: A comparative study of team formation in social networks, pp. 389–404, 2015. ISBN 978-3-319-18119-6. https://doi.org/10.1007/978-3-319-18120-2_23

Wang, X., Zhao, Z., Ng, W.: Ustf: a unified system of team formation. IEEE Trans. Big Data **2**(1), 70–84 (2016)

Wick, C.T.: The importance of team skills for software development. PhD thesis (1999)

Wieland, K., Langer, P., Seidl, M., Wimmer, M., Kappel, G.: Turning conflicts into collaboration. Comput. Supported Cooperative Work (CSCW) **22**(2), 181–240 (2013). https://doi.org/10.1007/s10606-012-9172-4

Xu, C., Sun, X., Li, B., Lu, X., Guo, H.: MULAPI: improving API method recommendation with API usage location. J. Syst. Softw. **142**, 195–205 (2018). https://doi.org/10.1016/j.jss.2018.04.060

Yang, H., Yan, Z., Jia, L., Liang, H.: The impact of team diversity on physician teams' performance in online health communities. Inf. Process. Manag. **58**(1), 102421 (2021)

Yasrab, R., Ferzund, J., Razzaq, S.: Challenges and issues in collaborative software developments (2011)

Ye, L., Sun, H., Wang, X., Wang, J.: Personalized Teammate Recommendation for Crowdsourced Software Developers, New York, NY, USA. Association for Computing Machinery, pp. 808–813 (2018). https://doi.org/10.1145/3238147.3240472

Zhang, Z., Sun, H., Zhang, H.: Developer recommendation for topcoder through a meta-learning based policy model. Empir. Softw. Eng. **25**(1), 859–889 (2020)

Zhu, H., Zhou, M., Seguin, P.: Supporting software development with roles. IEEE Trans. Syst. Man Cybern. Part A Syst. Hum. **36**(6), 1110–1123 (2006). https://doi.org/10.1109/TSMCA.2006.883170

Authors and Affiliations

Noppadol Assavakamhaenghan[1] · Waralee Tanaphantaruk[1] · Ponlakit Suwanworaboon[1] · Morakot Choetkiertikul[1] · Suppawong Tuarob[1]

✉ Suppawong Tuarob
suppawong.tua@mahidol.edu

Noppadol Assavakamhaenghan
noppadol.ass@student.mahidol.edu

Waralee Tanaphantaruk
waralee.tan@student.mahidol.edu

Ponlakit Suwanworaboon
ponlakit.suw@student.mahidol.edu

Morakot Choetkiertikul
morakot.cho@mahidol.edu

[1] Faculty of Information and Communication Technology, Mahidol University, Nakhon Pathom, Thailand