Empirical Software Engineering (2021) 26: 64
https://doi.org/10.1007/s10664-021-09966-4

Automatic team recommendation for collaborative software development

Suppawong Tuarob[1] · Noppadol Assavakamhaenghan[1] · Waralee Tanaphantaruk[1] · Ponlakit Suwanworaboon[1] · Saeed-Ul Hassan[2] · Morakot Choetkiertikul[1]

Accepted: 26 March 2021 / Published online: 7 May 2021

Abstract
In large-scale collaborative software development, building a team of software practitioners can be challenging, mainly due to overloading choices of candidate members to fill in each role. Furthermore, having to understand all members' diverse backgrounds, and anticipate team compatibility could significantly complicate and attenuate such a team formation process. Current solutions that aim to automatically suggest software practitioners for a task merely target particular roles, such as developers, reviewers, and integrators. While these existing approaches could alleviate issues presented by choice overloading, they fail to address team compatibility while members collaborate. In this paper, we propose *RECAST*, an intelligent recommendation system that suggests team configurations that satisfy not only the role requirements, but also the necessary technical skills and teamwork compatibility, given task description and a task assignee. Specifically, *RECAST* uses Max-Logit to intelligently enumerate and rank teams based on the team-fitness scores. Machine learning algorithms are adapted to generate a scoring function that learns from heterogenous features characterizing effective software teams in large-scale collaborative software development. *RECAST* is evaluated against a state-of-the-art team recommendation algorithm using three well-known open-source software project datasets. The evaluation results are promising, illustrating that our proposed method outperforms the baselines in terms of team recommendation with 646% improvement (MRR) using the exact-match evaluation protocol.

Keywords Team recommendation · Collaborative software development · Machine learning

Communicated by: Ali Ouni, David Lo, Xin Xia, Alexander Serebrenik and Christoph Treude

This article belongs to the Topical Collection: *Recommendation Systems for Software Engineering*

✉ Morakot Choetkiertikul
morakot.cho@mahidol.edu

Extended author information available on the last page of the article.

1 Introduction

Collaborative software development has become a new norm in the industry as software systems become larger and more complex, requiring not only teams of software practitioners with various technical backgrounds, but also a systematic and managerial approach to track progress and issues (Gharehyazie and Filkov 2017; Mistrík et al. 2010). To facilitate such processes, online software tracking tools have been developed, such as Jira, Backlog, and Bugzilla, that not only enable practical, systematic software development tracking, but also develop and encourage growing communities of software engineers and developers. Often, such communities of software practitioners can become so massive that it presents challenges for task assignees needing to choose potential team members to work on a particular software development task (e.g., building new features, resolving bugs, and integrating software).

Research has shown that ineffective software teams could lead to unsuccessful projects that result in a delay. In particular, assigning software development tasks to ineffective team members (e.g., developer, tester, and reviewer) may lead to low-quality outcomes and extra costs are required to rework (i.e., issue reopening) (Assavakamhaenghan et al. 2019). In a small community, assigning appropriate team members to complete software development tasks is trivial, as task assignees tend to be familiar with each person's technical background, relevant experience, and potential compatibility with other team members. However, manually building up a team in a large community of software practitioners with diverse technical backgrounds, experience, and social compatibility is challenging, calling for the ability to intelligently form and automatically recommend suitable teams that not only have the right skill set, but also are socially compatible with the other team members. This not only would assist decision-makers (e.g., task assignees, project managers, etc.) in choosing the right team members for the tasks, but also could be a key enabler in large-scale community-based collaborative software development ecosystems.

Recently, several recommendation methods have been proposed that automatically suggest software practitioners for a given task. However, such previous algorithms merely focus on making recommendations for particular common roles such as developers (Surian et al. 2011; Xia et al. 2015; Yang et al. 2016; Zhang et al. 2020b), peer reviewers (Zanjani et al. 2015; Thongtanunam et al. 2015; Ouni et al. 2016; Yu et al. 2016), and integrators (de Lima Júnior et al. 2018). An example of such single-role recommendation methods includes DevRec (Xia et al. 2013), a system for recommending developers for bug resolution. DevRec computes the affinity of each developer to a given bug report based on the characteristics of bug reports that have been previously fixed by the developer. While code bugs represent an aspect of software development tasks that can be handled by developers, a software task may involve diverse functional responsibilities such as gathering requirements, coding, reviewing, testing, integrating, and deploying, which require an ensemble of software practitioners each of whom is tasked with a distinct role. While one could use a set of different single-role recommenders to suggest potential members for each different role, such improvisation of single-role recommenders could overlook certain limitations, such as dynamic definitions and responsibility of roles, fine-grained skills, and anticipated teamwork compatibility (Gharehyazie et al. 2015; Moe et al. 2010).

While the above single-role recommenders have been proposed for software engineering tasks, each of which tends to be accomplished by one software practitioner, there are cases where a software task requires multiple experts with various technical backgrounds working together to reach a successful solution. For example, a task that aims to implement a stock-price forecasting functionality in a financial management mobile application may require a front-end Swift language developer, a machine learning engineer who can deploy TensorFlow models as services, a computer network programmer to engineer the data communication, and a code tester. This particular scenario would urge the task assignee to select team members that satisfy not only the necessary skills (i.e. Swift, TensorFlow, Python, and socket programming), but also the role requirements (i.e. three developers and one tester). The problem can become more aggravated if there are many choices of team members to choose from, which prevalently characterizes large open-source software systems, and could hinder the ability of the task leader to form an effective team. Such an issue, therefore, behooves an automated system that is capable of narrowing down the set of candidate software practitioners that also satisfy the skill and role requirements. In non-software engineering domains, systems capable of generating and recommending team configurations have been proposed in collaborative photography (Hupa et al. 2010), research collaboration (Datta et al. 2011; Liu et al. 2014), spatial crowdsourcing (Gao et al. 2017), business processes (Cabanillas et al. 2015), and collaborative learning (Ferreira et al. 2017), utilizing collaboration history, trust, and member skill information when deciding which teams to recommend. However, these techniques were not specifically developed for collaborative software development purposes, and hence cannot directly be applied to our problem.

A limited set of studies have investigated the ability to automatically recommend team configurations for collaborative software development. To the best of our knowledge, we are the first to explore this problem, and specifically investigate the following research questions:

1. Is it possible to quantify the ability of a software team to successfully resolve a given software development task?
2. Is it possible to recommend software teams that comply with the role requirements and are suitable for a given software development task?
3. Can the proposed software team recommendation method be adopted for single-role recommendation tasks?

Specifically, we propose *RECAST* (*REC*ommendation *A*lgorithm for *S*oftware *T*eams), a software team recommendation method for large-scale collaborative software development. The proposed method utilizes the Max-Logit algorithm to intelligently enumerate candidate teams to approximately maximize the team-fitness score at a task level (i.e., issue) since a resolving of an issue usually involves different roles and a number of team members, without having to exhaustively generate all possible combinations of teams. We propose a set of features that characterize effective software teams, given task requirements and task assignee (i.e. team leader). In this work, we define an effective team as a team that successfully resolves a task (e.g., an issue) without changing team members or re-opening the task. We also adopt features proposed by Liu et al. (2014) that capture the collaboration history and experience of team members. A set of machine learning algorithms are validated and used as the team-fitness scoring function. *RECAST* is validated on the ability to recommend both whole teams and particular single-roles. A state-of-the-art team recommendation algorithm proposed by Liu et al. (2014) along with a naive randomization-based baseline are used to validate the efficacy of our method on three real-world collaborative software project datasets: Apache, Atlassian, and Moodle, using the standard evaluation protocol for recommendation systems. According to the Mean Reciprocal Rank (MRR), the proposed *RECAST* achieves an improvement of 646% on average over the best baseline for the exact-match evaluation protocol. With such promising evaluation results, it would be possible to extend

RECAST to a real application that helps task assignees to find the right team configurations for their software tasks. Concretely, this paper presents the following key contributions:

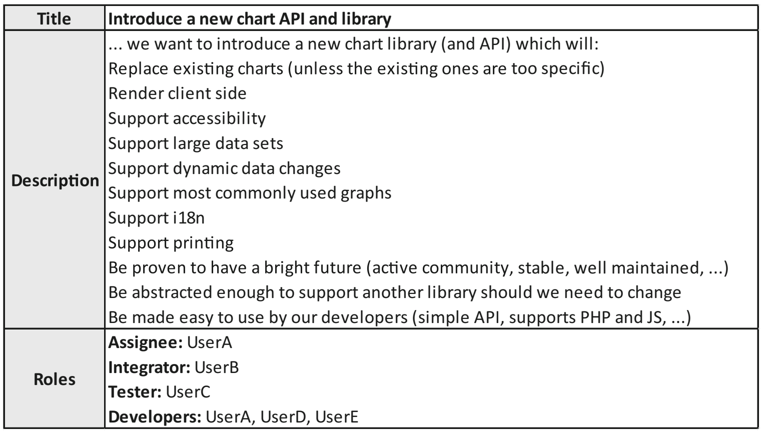
1. We establish the problem of automatic team recommendation in the software engineering context. While automatic team formation has been studied in the past, its applicability in collaborative software development has been limited.
2. We propose a learning-based method, *RECAST*, that recommends software teams, given a task description and assignee. The algorithm utilizes the Max-Logit algorithm to intelligently enumerate teams that approximately maximize the team-fitness score. A machine learning algorithm is used as the scoring function, which learns the features that characterize effective software teams. These features are derived from heterogeneous knowledge graphs that encode collaboration history, task similarity, and team members' skills, generated from real-world open source software project datasets.
3. We empirically validate our proposed team recommendation algorithm, *RECAST*, against a state-of-the-art team recommendation algorithm and a randomization-based baseline using the standard evaluation protocol for recommendation systems.
4. We make the datasets and source code available for research purposes at: https://github.com/NoppadolAssava/RECAST.

The remainder of the paper is organized as follows. Section 2 provides the mathematical definition and a real-world example of the problem. Section 3 explains relevant background concepts utilized by our proposed method. Section 4 provides the detail of our proposed method. Section 5 discusses datasets, experiment protocols, evaluation, and results. Section 6 exposes potential threads to validity. Section 7 reviews previous studies related to our problem. Section 8 concludes the paper.

2 Problem Description

Our goal is to develop an algorithm for recommending software teams for a given task and its role requirements. An example issue presented in Fig. 1, taken from the Moodle Tracker, is a software patch that introduces a new chart API and library to the current system. The usernames are anonymized. Carefully investigating the issue's comments and logs, we have identified four main roles, involving five team members. Note that, though there were more users involved in this issue, most of them were merely component watchers and non-active members (no activities); therefore, we only identified the active ones based on their comments and activities. Each of the team's active members was responsible for a different aspect of the task. For example, *UserA* was both the assignee and the main developer. *UserB* integrated the solutions. *UserC* created test cases and tested the solutions. *UserD* surveyed different chart plugins and found that Chart.js was the most suitable. *UserE* added a patch to the donut charts. This particular example illustrates the need for teamwork with diversified skills and responsibilities to accomplish a software task.

Mathematically, let $\mathbb{I} = \{i_1, i_2, i_3, ...\}$ be the set of tasks (i.e. issues), $\mathbb{U} = \{u_1, u_2, u_3, ...\}$ the set of users (potential team members), $\mathbb{R} = \{r_1, r_2, r_3, ...\}$ the set of roles, and $\mathbb{S} = \{s_1, s_2, s_3, ...\}$ the set of skills. Furthermore, we make a minimal assumption that each task, $i = \langle u_0, T, d\rangle$, is a triplet of a task assignee (i.e. $u_0 \in \mathbb{U}$), a team T, and a textual task description (i.e. d). A team $T = \{\langle r_1, u_1\rangle, \langle r_2, u_2\rangle, ...\}$ is a set of pairs of a role and its corresponding user. From the example Moodle issue in Fig. 1, the software team comprises



Title	**Introduce a new chart API and library**
Description	... we want to introduce a new chart library (and API) which will: Replace existing charts (unless the existing ones are too specific) Render client side Support accessibility Support large data sets Support dynamic data changes Support most commonly used graphs Support i18n Support printing Be proven to have a bright future (active community, stable, well maintained, ...) Be abstracted enough to support another library should we need to change Be made easy to use by our developers (simple API, supports PHP and JS, ...)
Roles	**Assignee:** UserA **Integrator:** UserB **Tester:** UserC **Developers:** UserA, UserD, UserE

Fig. 1 Example issue from the Moodle Tracker. Usernames are anonymized

one integrator: *UserB*, one tester: *UserC*, and three developers: *UserA*, *UserD*, and *UserE*. In this case, this team would be represented as follows:

$$T = \{\langle INT, UserB\rangle, \langle TEST, UserC\rangle, \langle DEV, UserA\rangle, \langle DEV, UserD\rangle, \langle DEV, UserE\rangle\}$$

Given a task i^* and assignee u_0, the objective of *RECAST* is to return a ranked list of K teams, $T(i^*) = \langle T_1, T_2, ..., T_K\rangle$ that best suite with the given task and role requirements, while ensuring technical skills and teamwork compatibility.

3 Background

This section presents relevant background concepts and knowledge. Developing a recommendation algorithm for software teams requires a presumption of collaborative software development processes, which can be tracked by software development tracking platforms such as Jira issue tracking system developed by Atlassian. Furthermore, since our proposed method uses sentiment analysis to extract features related to the social aspects of team collaboration, and Latent Dirichlet Allocation (LDA) to quantify the similarity between two tasks, we also provide an overview of such techniques.

3.1 Sentiment Analysis

Sentiment analysis refers to the use of natural language processing (NLP) and artificial intelligence techniques to automatically quantify subjectivity in the information. Specifically, sentiment analysis techniques extract meaningful subjective information, opinions, and emotions about the subject from written or spoken language. The primary function of sentiment analysis is to identify the sentiment polarity of a given text, namely positive, negative, or neutral. Advanced sentiment analysis techniques can understand emotional states such as happy, angry, and sad.

Typical sentiment analysis methodology includes five steps as shown in Fig. 2.

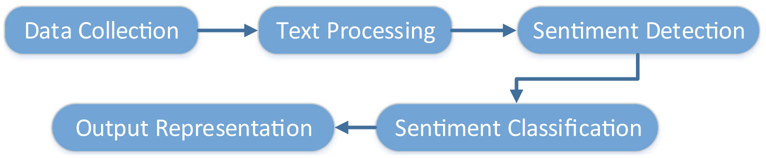


Fig. 2 Stages of the sentiment analysis process

1. **Data Collection**: Raw data (typically text) is collected from the sources. A unit of data, referred to as a document, may contain not only meaningful textual content, but also noises such as abbreviations, URL, slang, and meaningless, undefined words.
2. **Text Preparation**: Irrelevant information, non-text, and noises are eliminated. The raw text can be prepared by removing web addresses and URLs, stop words, symbols, and punctuation. This preparation step is different across data sources and target languages.
3. **Sentiment Detection**: The sentiment is detected from the input text. A document usually contains both objective statements describing facts and subjective statements that imply the author's sentiment. The sentiment detection algorithm then aims to identify those subjective statements to be used in the next step.
4. **Sentiment Classification**: The identified subjective statement presented in the document is classified into one of the sentiment classes (i.e. good/bad, positive/negative, etc.). Various techniques have been proposed to classify sentiment levels such as rule-based methods and machine learning based methods.
5. **Output Representation**: The last step is to present output in a meaningful way that fits with the applications (e.g. quantifying sentiment score in user reviews) (Baj-Rogowska 2017).

Teamwork collaboration often involves intra-communication among team members, which can be used to infer the team's coherence (Hogan and Thomas 2005). Studies have shown that ineffective communication could be an early indication of a premature team failure (Petkovic et al. 2014). Sentiment exhibited in communication can be signaling. For example, praising messages (positive sentiment) could lift the collaboration willingness among team members (Grigore and Rosenkranz 2011). Furthermore, messages that convey criticism (often detected as negative sentiment due to negative keywords) can enable the team to efficiently pinpoint the issues to be solved. The ability to mathematically extract such sentiment from past communication among members could be useful when choosing suitable teams for future tasks.

We conjecture that effective teamwork is characterized by not only the members' skills and experience but also a high-level of positive (e.g. praises and congratulations) and criticizing (e.g. critiques on features and bug reports) communication. Hence, sentiment analysis is used to identify the intent (as positive or criticizing) of the communication between two users in the same team, that is later used to compute the interaction-based features. Note that sentiment analysis is merely used to extract some "soft signal" in combination with other features to train machine learning models to characterize effective teams, and is not intended to be used as the absolute criteria to identify desirable software teams.

3.2 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) is an unsupervised topic modeling algorithm proposed and widely used in the information retrieval field (Blei et al. 2003). Specifically, LDA utilizes a probabilistic-based technique to model different topics from a set of documents, where each topic is mathematically represented by a probability distribution of terms. The inference algorithm allows a document to be represented with a probability distribution over different topics. The basic intuition behind LDA is that the author has a set of topics in mind while composing a document, which is represented as a mixture of topics. Mathematically, the LDA model is described as follows:

$$P\left(t_i|d\right) = \sum_{j=1}^{|Z|} P\left(t_i|z_i = j\right) \cdot P(z_i = j|d) \tag{1}$$

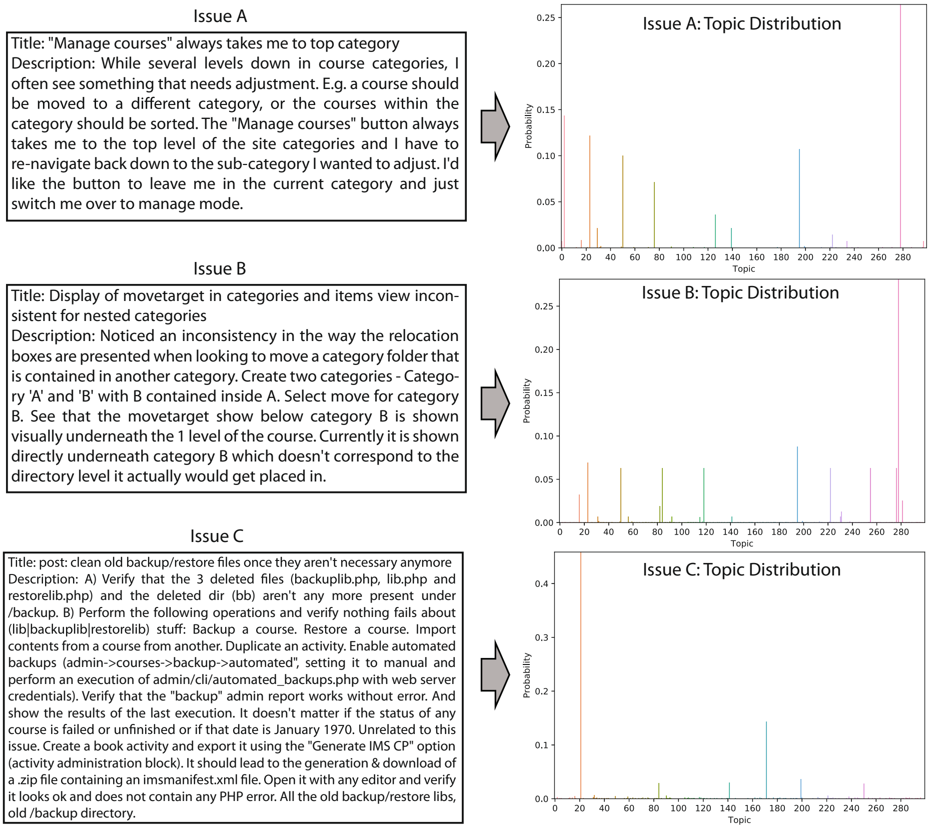
Where, $P(t_i|d)$ is the probability of term t_i appearing in document d, z_i a latent (hidden) topic, and $|Z|$ the number of all topics. Note that $|Z|$ needs to be predefined. $P(t_i|z_i = j)$ is the probability of the term t_i being in topic j. $P(z_i = j|d)$ is the probability of choosing a term from topic j in the document d.

After the topics are modeled, the distribution of topics can be computed for a given document using *statistical inference* (Asuncion et al. 2009). A document can then be represented with a vector of real numbers, each of which represents the probability of the document belonging to the corresponding topic.

LDA has been used in software engineering tasks in various applications such as analyzing the topical composition of repositories (Chen et al. 2016), logs (Li et al. 2018), and Stack Overflow articles (Rosen and Shihab 2016). One application of LDA is to measure the similarity between two documents, similar to the work by Al-Subaihin et al. (2019), who used LDA to quantify the similarity between mobile apps' textual descriptions. Particularly, two documents can be represented with topical distribution vectors where cosine can be used to measure the level of topical similarity between them. Note that, while simpler methods for quantifying document similarity exist such as Jaccard similarity and TF-IDF cosine similarity (Manning et al. 2008), studies have shown that LDA-based similarity yields better results particularly due to its ability to capture deep semantics compared to bag-of-words models such as TF-IDF and Jaccard based methods (Tuarob et al. 2015; Misra et al. 2008). Furthermore, projecting a document into a lower-dimensional space enables LDA to handle the polysemy, synonymy, and high-dimensionality problems that typically pose huge challenges for the bag-of-words based approaches (Alharbi et al. 2017).

In this research, LDA is used to quantify the similarity between two tasks (i.e., issues). Specifically, task descriptions are treated as textual documents from which LDA is first applied to learn the topics. To quantify the similarity between two tasks, their topic distribution vectors are then inferred from the learned topics. The cosine similarity between the two vectors (ranged [0, 1]) is then used as the level of similarity between the two tasks.

As an illustrative example, Fig. 3 shows the topic distributions of three Moodle's issues, namely A, B, and C. The corresponding textual content (title + description) of the three issues are displayed on the left-hand side, and the corresponding probability distributions of the 300 topics, inferred from a learned LDA model, are on the right-hand side. Visually, the distributions of Issue A and Issue B appear to be similar, with mutually high probabilities on Topics 23, 50, 195, and 278. However, the distribution of Issue C is much different from the other two. To interpret these results, even though Issues A and B are labeled as different components (*Administration* for Issue A and *Gradebook* for Issue B), they are both related to the user interface of the system that concerns the "course category" features (Issue A is an improvement, while Issue B is a bug). On the other hand, Issue C is about the backup mechanism of the system that is not quite related to Issues A and B. As a result, the cosine similarity of the topic distributions of Issues A and B is 0.79 (topically similar), while that



Issue A

Title: "Manage courses" always takes me to top category
Description: While several levels down in course categories, I often see something that needs adjustment. E.g. a course should be moved to a different category, or the courses within the category should be sorted. The "Manage courses" button always takes me to the top level of the site categories and I have to re-navigate back down to the sub-category I wanted to adjust. I'd like the button to leave me in the current category and just switch me over to manage mode.

Issue B

Title: Display of movetarget in categories and items view inconsistent for nested categories
Description: Noticed an inconsistency in the way the relocation boxes are presented when looking to move a category folder that is contained in another category. Create two categories - Category 'A' and 'B' with B contained inside A. Select move for category B. See that the movetarget show below category B is shown visually underneath the 1 level of the course. Currently it is shown directly underneath category B which doesn't correspond to the directory level it actually would get placed in.

Issue C

Title: post: clean old backup/restore files once they aren't necessary anymore
Description: A) Verify that the 3 deleted files (backuplib.php, lib.php and restorelib.php) and the deleted dir (bb) aren't any more present under /backup. B) Perform the following operations and verify nothing fails about (lib|backuplib|restorelib) stuff: Backup a course. Restore a course. Import contents from a course from another. Duplicate an activity. Enable automated backups (admin->courses->backup->automated", setting it to manual and perform an execution of admin/cli/automated_backups.php with web server credentials). Verify that the "backup" admin report works without error. And show the results of the last execution. It doesn't matter if the status of any course is failed or unfinished or if that date is January 1970. Unrelated to this issue. Create a book activity and export it using the "Generate IMS CP" option (activity administration block). It should lead to the generation & download of a .zip file containing an imsmanifest.xml file. Open it with any editor and verify it looks ok and does not contain any PHP error. All the old backup/restore libs, old /backup directory.

Fig. 3 Topic distributions inferred from the textual descriptions of three example Jira issues (i.e. Issues A, B, and C) taken from Moodle

of Issues A and C is only 0.01 (topically different), signifying that Issue B is more similar to Issue A than Issue C, which aligns with the interpretation from the textual content of the three issues.

Furthermore, Table 1 lists the top 10 topics ranked by the corresponding probabilities of Issues A, B, and C. The topic numbers with $*$ are those overlapped with the top 10 topics of Issue A. From this particular example, Issue B has five overlapped topics with those of Issue A, while Issue C has none. Generalizing from this particular example, if we could construct the topic distribution for all the tasks' textual description, then it would be possible to quantify the topical similarity between each pair of tasks using cosine similarity. Such a method for measuring document similarity has been used widely in the information retrieval domain. For example, Tuarob et al. (2015) used LDA to find similar documents to retrieve and recommend useful tags (keywords) in a document annotation task.

3.3 Potential Game Theory and Max-Logit algorithm

In the game theory literature, a potential game consists of a set of players and a set of possible actions for each player, with an assumption that there is one global *potential function* that represents all the players' incentive (Monderer and Shapley 1996). Each player then

Table 1 Top 10 topics and the corresponding probabilities of the sample three issues (i.e. A, B, and C)

Rank	Issue A		Issue B		Issue C	
	Topic	Probability	Topic	Probability	Topic	Probability
1	278	0.26425	**278***	0.28121	21	0.45862
2	2	0.14351	**195***	0.08778	171	0.14394
3	23	0.12167	**23***	0.06900	199	0.03620
4	195	0.10744	118	0.06285	141	0.02989
5	50	0.10004	84	0.06269	84	0.02954
6	76	0.07161	**222***	0.06268	250	0.02870
7	126	0.03595	**50***	0.06256	32	0.00709
8	139	0.02163	255	0.06254	90	0.00589
9	29	0.02150	276	0.06253	221	0.00554
10	222	0.01452	16	0.03241	16	0.00541

Topic numbers with $*$ are those that overlap with the top 10 topics of Issue A

chooses an action that increases the overall incentive. In every potential game, there exists at least one Nash equilibrium (Osborne and Rubinstein 1994), where a player no longer has an incentive to change the action. Such a concept can be applied to the team recommendation problem where each player is a role, and each player's action set is the set of candidate members for each role (Liu et al. 2014). At each iteration, a role changes its candidate to minimize a global cost function (equivalent to maximizing the potential function). This iterative process continues until the team reaches a Nash equilibrium.

Max-Logit (Song et al. 2011) is a Nash equilibrium finding algorithm, which can be used to simulate potential games. Compared to other Nash equilibrium finding algorithms, Max-Logit can not only escape from trivial local optima (using randomness), but also coverage to the best Nash equilibrium fast (Liu et al. 2014). In this research, since enumerating all possible team combinations can be exhausting due to the large candidate sets, characterizing large-scale software development communities, we use Max-Logit to enumerate teams that iteratively decrease the global cost function. The Max-Logit algorithm for team recommendation contains two major steps: constructing a new team and deciding whether to accept it as the best team (Liu et al. 2014). We have modified Max-Logit to fit with our software team recommendation which will be discussed in Section 4.

4 Methodology

Figure 4 illustrates the overview framework of *RECAST*. The data of software projects are collected from Jira issue trackers (see Section 4.1). The collected data are then prepossessed (e.g. removing incompleted issues). The common software issue-related (e.g. issue description) and software team-related (e.g. role) data are extracted from the preprocessed data (see Section 4.2). In this work, we also use heterogeneous knowledge graphs to capture relationship among users, tasks, and skills, which we discuss in Section 4.3. Several features are then extracted from these networks. Such features capture different aspects of team members' history of collaboration and task characteristics which are commonly used in team

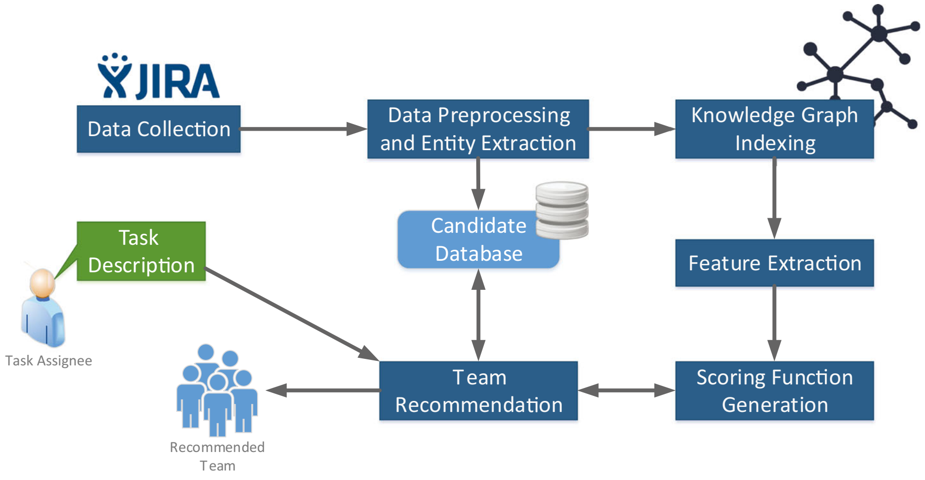


Fig. 4 Proposed method for recommending teams for collaborative software development

formation, such as technical experience, skill diversity, past collaboration among candidates (see Section 4.4).

We use machine learning techniques that learn the characteristics of "good" teams from past tasks, which will be used to quantify the effectiveness of a given team (see Section 4.5).

In the last step, to recommend software teams, our modified version of the Max-Logit algorithm is employed to enumerate candidate teams that maximize the team-fitness score. The top K candidate teams that achieve the highest team-fitness scores are returned as the recommended teams (see Section 4.6).

4.1 Data Collection

RECAST aims to recommend team combinations that are both tailored to the assigneees' preferences and contributing to the task's success. Hence, the knowledge graphs and the team-fitness scoring function must be generated and trained from real-world project-based software development data. Fortunately, most large-scale software projects are collaborative, whose development information is recorded in software project tracking tools such as Jira (issue tracking system),[1] Backlog (task tracking system),[2] and Bugzilla (bug tracking system).[3] Such software development tracking systems are often used by software practitioners to plan their projects, track progress, and to communicate among team members. Some tools such as Jira allow projects and tasks to have a hierarchical structure and dependency among them.

Real-world software project data is critical to train machine learning algorithms so that they can accurately predict the team combination suitable for a given task in the project. In this research, among different popular software tracking tools, Jira is chosen for our case studies due to the following reasons:

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

[2] https://backlog.com/

[3] https://www.bugzilla.org/

1. Open-source projects hosted by Jira can be publicly accessible on the Internet. This makes it possible to scrape public data using traditional web crawling techniques. Furthermore, Jira also provides a REST API (e.g., Moodle's API[4]) to facilitate information retrieval.
2. Jira has been used as the main software tracking tool for many open-source large-scale software systems such as Atlassian, Apache, and Moodle, which hosts several collaborative projects suitable for our case studies.
3. Jira has Github integration, which allows us to investigate commit activity logs. Such logs are crucial for identification of team members who are developers.

4.2 Data Preprocessing and Extraction

While Jira is chosen for our research, the raw data is preprocessed to retain only information that is not platform-specific. While different software tracking platforms have their own organizations of projects and tasks, here, we define a task to be a piece of collaborative work that is part of a software development. For example, Jira provides different features for different types of issues representing software development tasks, such as new features, bugs, and implementation.

We assume that a task is composed of a task description, a task assignee, and a team. Furthermore, additional information such as a task resolution and interaction among team members is also collected.

4.2.1 Task Description

A task description is usually represented as a simple text document. Hence, any attributes that describe a task are concatenated to produce a continuous chunk of text. For example, Jira provides the following features to describe an issue: summary and description. The textual contents corresponding to these attributes would be concatenated to produce the task's description.

4.2.2 Task Status Resolution

RECAST's goal is to recommend teams that are not only tailored for the task assignee's preference, but also functionally effective. Hence, the training data used to train our models must come from tasks that are deemed successful. In this research, a *successful* task is a task (issue) that is resolved without being re-opened. In Jira, this means that a successful issue must have status "Closed" and resolution "Resolved" without any history of reopening. Furthermore, we track whether a task has been reopened from the issue changelog. Issues that do not meet such a definition are discarded from the training set.

Note that, while this definition of a *successful* task given above can be subjective, a study has shown that tasks that have been re-opened are likely to be caused by ineffective team configurations (Assavakamhaenghan et al. 2019). Hence, to ensure that the models learn from high-quality samples (i.e. teams that lead to resolving of issues), only tasks that can be resolved without any major interruption that may have caused by flawed teams are used as the training dataset. Such a definition of a successful task only has a direct effect on the generation of the team-fitness scoring function. Hence, if future studies devise a different

[4]https://tracker.moodle.org/rest/api/2/issue/issueid

definition of a “good team,” then the training data can be relabeled without affecting the rest of the methodology pipeline.

4.2.3 Task Dependency

Most modern software tracking tools provide the capability for users to specify explicit relations among tasks. For example, Jira provides the “*Issue Link*” mechanism so that the relation between two issues can be explicitly and systematically specified. Examples of such issue relations include *Relates to*, *Duplicates*, *Blocks*, and *Clones*. In this research, to align with our generalized assumption about task hierarchical structure, such issue links are collected to represent the dependency relations between two tasks.

4.2.4 Team Members

Besides the task assignee, we also collect the username, and role of each team member working on the task. Jira focuses on the implementation aspect of the software, the following roles related to issue resolving can be identified: Reporter, Assignee, Developer, Tester, Peer Reviewer, and Integrator.

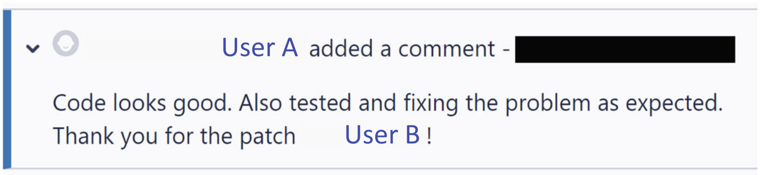
4.2.5 User Interaction

Social aspects are involved throughout the collaborative development of software, which can be reflected by communication. In this research, we investigate features that characterize team coherence (i.e., the ability for a team to productively work together). Many studies have shown that communication and teamwork are among the key factors for successful software projects (Sudhakar 2012; Lindsjørn et al. 2016).

In this research, communication information among team members is collected from the comment on issue reports. Specifically, only direct comments from one member to others are collected. An example of a direct comment from the Moodle project is shown in Fig. 5, where `User A`, the peer reviewer, shows gratitude to `User B`, the assignee, after finishing the code review. While this particular example alone may not be enough to conclude that these two members exhibit a strong characteristic of good teamwork, but hundreds of such direct messages between the two users from several previous issues could be revealing.

4.2.6 Sentiment Analysis

In every issue, team members communicate with each other using comments. For example, they can use the comments to praise for milestones reached or criticize others’ work. These comments are full of sentiment that could be signaling. Therefore, all comments are analyzed to extract sentiment using *SentiStrength*, a python library for sentiment analysis


User A added a comment - [redacted]

Code looks good. Also tested and fixing the problem as expected. Thank you for the patch User B !

Fig. 5 Example of a direct comment from a Moodle issue (Usernames are anonymized)

(Thelwall et al. 2010).[5] Specifically, the following steps are taken to extract sentiment from a comment:

1. **Text Preprocessing**: All comments are cleaned by removing stop-words, punctuation, URLs, white-spaces, and numbers. Special symbols used to represent *emoji* are not removed to preserve emotional expression. A textual comment is then tokenized using the NLTK tokenization tool.
2. **Sentiment Detection**: SentiStrength library outputs two values representing positive and negative sentiment. The reason for having the two sentiment scores instead of just one (with –/+ sign representing negative/positive sentiment) is because research findings have determined that positive and negative sentiments can coexist (Fox 2008). Note that, from our observation, while positive comments are those related to praises and thanks, the negative ones are not from hate-speech or ill-language as traditionally interpreted from the negative scores. Often, negative comments tend to express criticism on software functionalities or to explain bugs (which necessitate negative words), rather than expressing the authors' emotion.
3. **Trust Propagation**: According to Kale et al. (2007), trust and distrust can be propagated from one person to another in the person-person network. In their work, they quantified trust and distrust from the text surrounding each URL link in a blog, that leads to another user's blog. They computed the link sentiment polarity by quantifying the sentiment of the text surrounding it. The positive sentiment represents trust and the negative sentiment represents distrust. They spread trust links to increase the density of the network so that they could detect trust communities in the network.

   Similarly, in our work, interaction among users is not dense enough. Hence, we adopt Kale's idea to increase link pairs of interaction. However, we do not define positive sentiment as trust and criticism sentiment as distrust. This is because, in software development, a comment containing negative words usually criticizes software features so that developers could effectively understand the requirements. However, we could use Kale's trust propagation model to propagate positivity and criticism within the interaction network. Mathematically, link polarity can be propagated using this following matrix operation:

   $$C = \alpha_1 B + \alpha_2 B^T B + \alpha_3 B^T + \alpha_4 B B^T \tag{2}$$

   Where C is an atomic propagation operator, B is the initial belief between user i and j. For α vector, they reported that {0.4, 0.4, 0.1, 0.1} yields the most accurate result. The belief B can be computed iteratively depending on the steps of propagation $i + 1^{th}$, B_{i+1}, using the following equation:

   $$B_{i+1} = B_i * C_i \tag{3}$$

4.3 Network Generation and Indexing

Effective software teams are typically characterized by having extensive experience of relevant tasks, necessary skills, a good history of working together. Hence, the ability to automatically quantify such characteristics could prove useful to a software team recommendation system. Here, we conjecture that such team-related information can be extracted from the heterogeneous networks that represent the relationship between past tasks, users,

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

and skills. The following subsections define different types of information networks used in this research, along with explaining how they are generated from the collected dataset.

4.3.1 Task Similarity Network

The task similarity network $\mathbb{G}_{TS} = \langle \mathbb{I}, E_{TS} \rangle$ is an undirected graph where a node $i \in \mathbb{I}$ is a task, and each edge weight $e \in E_{TS}$ represents the similarity between the two tasks. We assume that an issue description is represented with a textual document without any available ontology; hence, the task similarity must be quantified using the textual content derived from the issue descriptions.

In this research, task (i.e. issue) descriptions are represented with topical distribution inferred from trained Latent Dirichlet Allocation (LDA) models. Both the traditional non-labeled LDA and labeled LDA (Ramage et al. 2009) are validated for their ability to model latent topics for each corpus of task descriptions. We use Mallet[6] implementation of these LDA algorithms. The number of topics is tuned to optimize the local pairwise mutual information (PMI). The similarity between the two tasks can be calculated using the cosine similarity between the two tasks' topical distribution vectors. Hence, each edge weight falls into the range [0,1] where 0 represents no similarity and 1 represents perfect similarity.

4.3.2 Task Dependency Network

The task dependency network $\mathbb{G}_{TD} = \langle \mathbb{I}, E_{TD} \rangle$ is a heterogeneous directed graph, where each node $i \in \mathbb{I}$ is a task, and each direct edge $e(i_x, i_y, r)$ represents that task i_x depends on task i_y with type r, where the type of this edge can be either *Relates*, *Blocks*, or *Clones*. The dependency relationship between two tasks is extracted from the "Issue Link(s)" attribute in a Jira issue page.

4.3.3 User Collaboration Network

The user collaboration network $\mathbb{G}_{UC} = \langle \mathbb{U}, E_{UC} \rangle$ is an undirected network where each node is a user $u \in \mathbb{U}$ and each edge weight represents the frequency of tasks on which the two users have collaborated. In Jira, the collaboration relationship is directly extracted from the issue information page where all the collaborators are listed. Specifically, two users collaborate if they are identified as the team members of the same issue. This user collaboration network allows us to understand the collaboration history of the members in a given team, and also enables quantification of team coherence.

4.3.4 User Interaction Network

History of interaction among users can be signaling and helpful for understanding the dynamics within a software team. Ideally, an interaction can be directly sending personal messages to one or more other team members. However, since direct message mechanisms are not available in Jira, we treat a direct comment posted in an issue page (See example in Fig. 5) as an interaction between the message poster and the mentioned users.

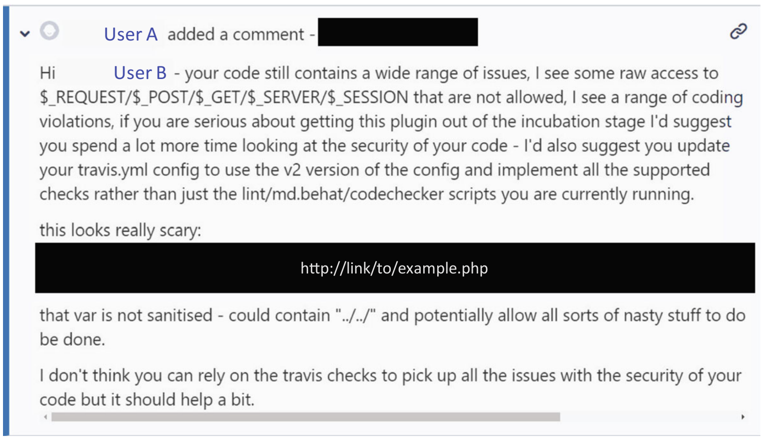
[6]http://mallet.cs.umass.edu/topics.php

Emotions play a big role in teamwork. We observe that effective teams are characterized not only by a high level of communication, but also by meaningful and constructive interaction. Specifically, meaningful comments such as praises, encouragements, and congratulations can raise ones' motivation and morale. Also, constructive feedback about ones' work can lead to efficient and productive improvement and resolution. For example, in Fig. 6, *User A* made a direct comment to *User B* stating that his code still contains certain issues, with detailed explanation. As a result, *User B* knew exactly what to proceed to address these comments, and was able to resolve and close this task. While these constructive feedback comments may sound professional and do not intend to offend the target user, the sentiment quantified by traditional sentiment analysis tools is often negative. This is due to the composition of negative keywords used to describe components in the code, such as `issues`, `serious`, and `scary`. Note that, in fact, due to the professional and mature community of software developers, we have only witnessed a few negative comments (as reported by the sentiment analysis tool) that turn out to be offensive or showing that the authors had negative sentiment when writing the comments. Hence, we assume that positive comments are those intended to praise, encourage, or congratulate team members, while negative ones are typically constructive feedback.

The user interaction network $\mathbb{G}_{UI} = \langle \mathbb{U}, E_{UI} \rangle$ hence is a direct heterogeneous graph where each node is a user $u \in \mathbb{U}$, and each edge $e(u_i, u_j, polarity)$ represents the direction of communication (i.e. from u_i to u_j), and the type of this edge $polarity$ can be either positive (+) or negative (-) sentiment. The weight of each edge is the sentiment level, ranging from [0,1]. Note that, we keep both the positive and negative edges, instead of combining the positive and negative weights into one single value, because our work interprets negative sentiment differently from its original definition. Specifically, here we treat both positive and negative comments to be good characteristics of effective teams.

4.3.5 User Expertise Network

Forming a team having not only necessary technical skills, but also diverse expertise is crucial to the task's success. The user expertise network captures the skill set that each user



User A added a comment - [redacted]

Hi User B - your code still contains a wide range of issues, I see some raw access to $_REQUEST/$_POST/$_GET/$_SERVER/$_SESSION that are not allowed, I see a range of coding violations, if you are serious about getting this plugin out of the incubation stage I'd suggest you spend a lot more time looking at the security of your code - I'd also suggest you update your travis.yml config to use the v2 version of the config and implement all the supported checks rather than just the lint/md.behat/codechecker scripts you are currently running.

this looks really scary:

http://link/to/example.php

that var is not sanitised - could contain "../../" and potentially allow all sorts of nasty stuff to do be done.

I don't think you can rely on the travis checks to pick up all the issues with the security of your code but it should help a bit.

Fig. 6 Example constructive feedback. Usernames and sensitive information are anonymized

possesses based on his/her past task experience. Mathematically, the user expertise network is a bipartite direct graph $\mathbb{G}_{UE} = \langle \mathbb{U}, \mathbb{S}, E_{UE} \rangle$ where a node can be either a user $u \in \mathbb{U}$ or a skill $s \in \mathbb{S}$, and the weight of each edge $e(u, s)$ represents the level of skill s that user u possesses. In Jira, there is no direct way to retrieve explicit software skills (e.g. Java, Python, Deep Learning, etc.) of a particular user. Hence, we use the system components (e.g. Checklist, TurnItIn tool, Application Form) to represent the set of skills. The reason behind this is that each software system already has a well-defined set of system components. A user who has worked on particular components are said to have acquired the necessary skills and experience on them. The weight of each edge $e(u, s)$ is then quantified by the frequency of the tasks related to component s whose teams comprise user u.

4.4 Feature Extraction

The proposed method relies on a machine learning based scoring function that outputs the team-fitness score given task description, a task assignee, and a candidate team. To build such a scoring function, machine learning based algorithms are trained with features extracted from teams whose tasks are deemed successful. Specifically, these features are designed to characterize teams that are suitable for a given task and task assignee. A majority of the features are extracted from the heterogeneous knowledge networks discussed in Section 4.3. We also incorporate features proposed by Liu et al. (2014), as they can be additionally useful. Table 2 lists all the features.

Table 2 List of features used to train the team-fitness scoring functions

Network	Feature
Task Similarity	Task Familiarity
Task Dependency	Task Proximity
User Collaboration	Relatedness to Assignee
	Team Coherence
User Interaction	Team Positivity
	Team Criticism
	Positivity towards Assignee
	Criticism towards Assignee
User Expertise	Skill Competency
	Skill Diversity
	Component Experience
Other	Team Contribution
	Domain Experience
	Co-Task Frequency
Liu et al.	Experience
	Role Experience
	Closeness
	Success Experience
	Connection
	Success Rate

4.4.1 Features Adopted from (Liu et al. 2014)

Liu et al. (2014) proposed a recommendation algorithm for scientific collaboration. In this work, we find that the features used in their work could be useful to our problem. These previously proposed features include:

1. **Experience** reflects the team's experience in handling overall tasks. We use the number of tasks in which a member has involved to reflect experience. This feature is normalized with Min-Max normalization. To calculate the overall team experience, the average is used.
2. **Role Experience** captures the team's experience in different roles. We use the number of tasks in which a person participated in a particular role to compute this feature. This feature is also normalized using Min-Max normalization. To calculate the overall role experience of the team, the average is used.
3. **Win Experience** represents overall experience with successful tasks. This feature is computed using the number of successfully resolved issues that a person participated in, normalized with the Min-Max normalization. To calculate the overall win experience of the team, the average is used.
4. **Win Rate** is the ratio of `Win Experience` to `Experience`. To calculate the overall experience of the team, the average is used.
5. **Closeness** quantifies the overall collaboration history among the members in a team T, and is calculated using the equation below (Liu et al. 2014).

$$Closeness = \frac{2}{|T| \times (|T|-1)} \sum_{u_i,u_j \in T; i<j} \frac{1}{ShortestPath(u_i, u_j)} \tag{4}$$

   $|T|$ is the cardinality of the team. The $ShortestPath$ is calculated from the graph $\mathbb{G}_{UC} = (\mathbb{U}, E_{UC})$ where vertices $\mathbb{U}$ is the set of users. An edge $e_i \in E_{UC}$ links $u_i \in \mathbb{U}$ and $u_j \in \mathbb{U}$ with weight representing the number of tasks u_i and u_j have worked together. If there is no path between u_i and u_j, $|\mathbb{U}|$ is used instead.
6. **Connection** represents overall level of the team's communication activities, and is described as follows:

$$Connection = \frac{2}{|T| \times (|T|-1)} \sum_{u_i,u_j \in T; i<j} e_{ij} \tag{5}$$

   Liu et al. (2014) represents e_{ij} with the number of connections between u_i and u_j. In our research, we use the number of interactions (tagged users in comments) between u_i and u_j, derived from the user interaction graph $\mathbb{G}_{UI}$.

4.4.2 Proposed Features

While features proposed by Liu et al. (2014), described in the previous section, can be useful for characterizing the team's overall experience and collaboration. We find that they are not sufficient to capture certain characteristics of teams that work in software projects, including task requirements, skills, and collaborative compatibility; hence, we define additional software development specific features as follows:

1. **Task Familiarity** indicates the team's experience dealing with similar tasks in the past, derived from the task similarity graph $\mathbb{G}_{TS}$, where each task is represented as a vector of topic distribution inferred from a trained topic model using LDA. The

similarity between two tasks is then computed using the cosine similarity of their topic distribution vectors. Mathematically, the *Task Familiarity* is calculated as follows:

$$Task\ Familiarity = \frac{1}{|T|}\sum_{u \in T} max_{i \in I_u}\left(cosinesim\left(i^*, i\right)\right) \tag{6}$$

Where T is the set of team members, u is a user, i^* is the target task, I_u is the set of tasks in which u was involved.

2. **Task Proximity** reflects the team's overall experience working in closely related tasks and is derived from the task dependency network $\mathbb{G}_{TD}$. It is calculated from the inverse of the average of the minimum shortest path between each pair of tasks done between the task assignee and other team members. Note that we only compute the task proximity between the task assignee and each team member, instead of between every possible pair of team members, because we would like this feature to capture team members working in closely related tasks that could be beneficial to the target task initiated by the task assignee. Furthermore, it is often that a person may be involved in more than 1,000 tasks. Calculating the shortest path between all of this user's tasks to others' makes it extremely computationally expensive. While one may argue that this feature might be better captured between the target task i^* (instead of the assignee's past tasks) and the team members' tasks, in practice, the task dependency is not always established during the task initiation, but during the task execution. Hence, we do not assume that such task dependency information between the target task and the previous tasks is available. Concretely, the task proximity feature is computed as follows:

$$Task\ Proximity = \frac{|T| \times (|T| - 1)}{\alpha + \sum_{u \in T, u \neg u_0} min_{i_{u_0} \in I_{u_0}, i_u \in I_u}(ShortestPath_N(i_{u_0}, i_u))} \tag{7}$$

Where T is the set of team members, u is a member, u_0 is the task assignee, i is a task, I_u is the set of tasks that u was involved, $ShortestPath_N(i_{u_0}, i_u)$ is the shortest path between the tasks involved by u_0 and u with the maximum length of N, α is the smoothing factor to prevent the zero-division error. In our research, we use $\alpha = 1$ and $N = 1,000$.

3. **Team Coherence** reflects how the collaboration history of the team members, derived from the user collaboration graph $\mathbb{G}_{UC}$. It is calculated using the inverse of the average shortest path between each pair of members. A longer path implies less coherence.

$$Team\ Coherence = \frac{|T| \times (|T| - 1)}{\alpha + \sum_{u_i, u_j \in T; i<j}(ShortestPath_N(u_i, u_j))} \tag{8}$$

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

4. **Relatedness to Task Assignee** ($Relatedness_{u_0}$) is similar to team coherence. However, this feature focuses on the coherence between the task assignee and the rest of the team.

$$Relatedness_{u_0} = \frac{|T| \times (|T| - 1)}{\alpha + \sum_{u \in T}(ShortestPath_N(u_0, u))} \tag{9}$$

Where T is the set of team members, u is a member, $ShortestPath_N(u_0, u)$ is the shortest path from the task assignee u_0 to each member u with the maximum of N, α is the smoothing factor to prevent the zero-division error. In our research, we use $\alpha = 1$ and $N = 1,000$.

5. **Team Positivity** reflects how team members interact with positivity in the past, derived from the user interaction network $\mathbb{G}_{UI}$, considering only positive edges. We adopt trust propagation method (Kale et al. 2007) to expand the relationship of pairs as previously discussed in Section 4.2.6.

$$Team\ Positivity = \frac{1}{|T| \times (|T|-1)} \sum_{u_i,u_j \in T; i<j} I^{+}(u_i, u_j) \tag{10}$$

   Where T is the set of team members, u_i and u_j are a pair of team members, $I^{+}(u_i, u_j)$ is the average weight of the positive edges between u_i and u_j in the user interaction graph $\mathbb{G}_{UI}$ after applying the trust propagation model.

6. **Team Criticism** reflects how team members express constructive criticism towards each other. It is derived the same way as the team positivity discussed previously, but only uses the negative edges in the user interaction network.

$$Team\ Criticism = \frac{1}{|T| \times (|T|-1)} \sum_{u_i,u_j \in T; i<j} I^{-}(u_i, u_j) \tag{11}$$

   Where T is the set of team members, u_i and u_j are a pair of team members, $I^{+}(u_i, u_j)$ is the average weight of the negative edge weights between u_i and u_j after applying the trust propagation.

7. **Positivity towards Task Assignee** is similar to *Team Positivity*, but only considers the positivity (e.g., praises, gratitude, and congratulations) expressed by team members towards the task assignees.

$$Positivity \rightarrow Task\ Assignee = \frac{1}{|T|} \sum_{u \in T} I^{+}(u, u_0) \tag{12}$$

   Where T is the set of team members, u is a team member, u_0 is the task assignee, $I^{+}(u, u_0)$ is positive interaction score from u to u_0.

8. **Criticism towards Task Assignee** is similar to *Team Criticism*, but only focuses on the criticism made towards the task assignee.

$$Criticism \rightarrow Task\ Assignee = \frac{1}{|T|} \sum_{u \in T} I^{-}(u, u_0) \tag{13}$$

   Where T is the set of team members, u is a team member, u_0 is the task assignee, $I^{-}(u, u_0)$ is negative interaction score from u to u_0.

9. **Skill Competency** quantifies the team's ability (skills) to satisfy the target task's requirements. Each member's skills are derived from the user expertise network $\mathbb{G}_{UE}$. Mathematically, this feature is computed using a set-overlap between the team's skills and the required skills.

$$Skill\ Competency = \frac{|S_T \cap S_r|}{|S_r|} \tag{14}$$

   Where S_T is the set of team members' skills, S_r is the set of required skills.

10. **Skill Diversity** reflects the diversity of the team members. Teams with high skill diversity tend to comprise individuals who are experts in particular skills, rather than a few persons who have abundant but shallow skills. Hence, this feature, when working together with the skill competency, can distinguish teams comprising experts with necessary skills, and prevent the scoring function from giving high scores to teams comprising only a few persons who can do a little of everything. Skill diversity is

measured by the number of unique, required skills that a team can offer (no duplicate skills), divided by the sum of required skills offered by each team member.

$$Skill\ Diversity = \frac{|S_T \cap S_r|}{\sum_{u \in T} |S_u \cap S_r|} \tag{15}$$

Where T is the set of team members, S_T is the set of team skills, S_r is the set of required skills, S_u is the skill set of u.

11. **Skill Experience** computes the experience of the team with respect to each required skill. It considers each member's past tasks that require the same set of skills as those of the target task.

$$Skill\ Experience = \frac{1}{|T||S_r|} \sum_{u \in T} \sum_{s \in S_u \cap S_r} |P_u(s)| \tag{16}$$

Where T is the set of team members, u is a user, S_r is the set of required skills and S_u is the set of skills that u experienced, $P_u(s)$ is the set of tasks that require skill s and involve user u.

12. **Team Contribution** directly quantifies the team activeness, considering the collective history of the team members' activities in the past tasks. Activities include changing issue status, making comments, and updating code to the repository, which can be collected from each tasks' log (i.e. a changelog of an issue report). The contribution of a member is defined as the average ratio of the member's activities in his/her previous tasks.

$$Team\ Contribution = \frac{1}{|T|} \sum_{u \in T} \frac{\sum_{i \subset I_u} c_{u,i}}{|I_u|} \tag{17}$$

Where T is the set of team members, u is a user, I_u is a set of tasks in which u participated, $c_{u,i}$ is the contribution of member u in task i which can be calculated using the following formula:

$$c_{u,i} = \frac{|Activities\ of\ u\ in\ task\ i|}{|All\ activities\ in\ task\ i|} \tag{18}$$

13. **Domain Experience** reflects team members' experience on the tasks within the same domain of the target task. This is different from the *Experience* feature discussed in Section 4.4.1 which quantifies the team experience from all the past tasks. Domain experience can be calculated if the software tracking platforms allow tasks to have a hierarchical structure. In Jira, a software system is divided into different domains (i.e. Jira projects), each of which is used to categorize tasks into groups.

$$Domain\ Experience = \frac{1}{|T|} \sum_{u \in T} |D_{p*}(u)| \tag{19}$$

Where T is the set of team members, u is a user, $p*$ is the target task, $D_{i*}(u)$ is the set of tasks in the same domain as i^* and involve u.

14. **Co-Task Frequency** reflects how often the team members work together, derived from the average number of tasks co-worked by each pair of team members.

$$Co - Task\ Frequency = \frac{1}{|T| \times (|T| - 1)} \sum_{u_x, u_y \in T; x<y} |I_{u_x, u_y}| \tag{20}$$

Where T is the set of team members, u is a user, I_{u_x,u_y} is the set of tasks that u_x and u_y work together.

4.5 Scoring Function Generation

This section describes the method for choosing the best machine learning algorithms to use as the *team-fitness* scoring function. The function is used to compute scores for candidate teams, allowing them to be ranked. Mathematically, the team-fitness scoring function $F_{fitness}(d, u_0, t) \in [0, 1]$ quantifies the chance that a given team t would be a suitable team for a task described by d and led by user u_0.

We frame the team-fitness quantification into a binary classification problem, where a classifier predicts a given team whether it is effective or non-effective. An effective team is defined as a team that leads to task success. Therefore, for a given task, if the corresponding team completes the task with success (the definition of a successful task is defined in Section 4.2.2), such a team is considered effective. The probability output from such a classifier is then used as the team-fitness score. To train the scoring function, teams from successful tasks (See Section 5.1) are treated as positive samples. For each positive sample, 100 negative samples are synthesized by filling a random (role-compatible) user in each role. Such a data labeling method would allow the classifier to better recognize the characteristics of teams that lead to task success. For each task, the task description, task assignee, and the team members are extracted and used to compute team features discussed in Section 4.4.2.

A number of machine learning classification models drawn from diverse families of classification algorithms are considered, including Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Naive Bayes (NB), Quadratic Discriminant Analysis (QDA), k-Nearest Neighbors (kNN), and Artificial Neural Network (ANN). These traditional machine learning classification models have been widely used to validate the predictability of many classification tasks in many reputable papers (Tuarob et al. 2020; Tantithamthavorn et al. 2018; Jiarpakdee et al. 2020; McIntosh et al. 2019; Choetkiertikul et al. 2017). Regardless, the proposed framework is highly configurable in the sense that other classifiers could be tested without much change to the existing code.

5-Fold cross-validation is used to validate each model using ROC as the main criteria. The best model for each dataset is then chosen to be the team-fitness scoring function and will be used together with the Max-Logit algorithm to recommend teams.

4.6 Team Recommendation

To recommend software teams, *RECAST* takes a task description, task assignee, and the required roles as inputs, and output a list of K recommended teams, ranked by the team-fitness scores. Here, the variable K is configurable. A naive method would be to enumerate all the possible candidate teams from the given roles, compute the team-fitness score for each candidate team, and then rank all the teams based on the assigned scores. However, the size of all the possible team combinations can become exponentially large, especially for large-scale software communities that house thousands of software practitioners. To intelligently select a subset of all the possible combinations of teams to compute the team-fitness scores, we adopt the Max-Logit algorithm (Dai et al. 2015). Such a Nash-equilibrium finding algorithm narrows down the subset of candidate teams by iteratively finding a better team after each iteration. Max-Logit has also been used by Liu et al. (2014) to prune the search space for team recommendation. However, according to Liu et al. (2014) the original Max-Logit implementation was designed to output only one approximately best team after the algorithm terminates.

In this work, since we would like to recommend more than one candidate team, the original Max-Logit algorithm is modified to also keep track of enumerated teams and their corresponding scores, so that these teams can be part of the recommendation, as outlined in Algorithm 1. In Line 8, the newly enumerated teams and their costs are retained in $RecTeams$, so that they can be sorted, selected, and returned in Lines 15 and 16. The cost function $Cost(T)$ is used to compute the cost of a given candidate team T, and is defined as the inverse of the team-fitness score. Interested readers are encouraged to consult the original definition of the Max-Logit algorithm by Monderer and Shapley (1996).

Algorithm 1 Modified max-Logit for TopK team recommendation.

```
Input  : 1. A task with a set of required roles
         2. A set of candidates for each role
         3. Function $Cost(T)$ that computes the cost of a given team T
         4. Number of iterations $N$
         5. Smoothing factor $\tau$
         6. Number of recommended teams $K$
Output: Top K best teams and their costs
1  $RecTeams$ = array()
2  Randomly select candidate for each role and generate a team $T$
3  Append $(T, Cost(T))$ to $RecTeam$
4  for $i = 1$ to $N$ do
5      Calculate $Cost(T)$
6      $T' \leftarrow$ Randomly select a role in $T$, and replace its user with a randomly selected
       candidate, get a new team
7      Calculate $Cost(T')$
8      Append $(T', Cost(T'))$ to $RecTeams$
9      $prob \leftarrow PROBABILITY(Cost(T), Cost(T'))$
10     $r \leftarrow random(0, 1)$
11     if $r \leq prob$ then
12         $T \leftarrow T'$
13     end
14 end
15 Sort $RecTeams$ by $costs$ (ascending)
16 $RecTeams \leftarrow RecTeams[: K]$
17 return $RecTeams$
18
19 Function $PROBABILITY(Cost(T), Cost(T'))$
20     $v_t = exp^{-Cost(T)/\tau}$
21     $v_{t'} = exp^{-Cost(T')/\tau}$
22     $prob = \frac{v_{T'}}{max(v_t, v_{T'})}$
23     return $prob$
24 end
```

4.7 Evaluation Metrics for Team Recommendation

Precision@K, Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), Mean Rank (MR), and Mean Rank of Hits (Hit MR) are used as the evaluation metrics. These metrics, when used in combination, have been shown effective for evaluation of recommendation systems (Zhang et al. 2020b; Ragkhitwetsagul and Krinke 2019; Tuarob et al. 2015). Let

Q be the set of test tasks, i.e. the query set, $T(p \in Q) = \langle T_1, T_2, ..., T_K \rangle$ be the ranked list of recommended K teams for task p, T_0 be the actual team, and $H(T_0, T_i)$ be a binary function that returns 1 if T_i is a match with T_0 (i.e. a hit), and 0 (i.e. a miss) otherwise. In this research, two definitions of $H(T_0, T_i)$ is used:

1. **Exact-Match** ($H_{EXACT}(T_0, T_i)$): T_i is said to be an exact-match with T_0 if the set of team members in T_i is the same as T_0.
2. **Half-Match** ($H_{HALF}(T_0, T_i)$): T_i is said to be a half-match with T_0 if at least half of the team members of T_i overlap with those of T_0.

The evaluation metrics are then defined as:

4.7.1 Precision@K and Mean Average Precision (MAP)

Precision is a well-known metric in the information retrieval literature, traditionally used to quantify how precise the predicted answers are. Traditional precision does not take ordering of correct answers into account. Hence, for the recommendation task, precision is computed at every top K recommendations to justify the optimal K that maximizes the tradeoff between preciseness and variety of the recommended results.

$$Precision@K = \frac{\sum_{i=1}^{K} H(t_0, t_i)}{K} \tag{21}$$

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.

$$MAP = \frac{\sum_{i=1}^{10} Precision@i}{10} \tag{22}$$

4.7.2 Mean Reciprocal Rank (MRR)

MRR takes the order of the first correct answers into account and is a reliable accuracy-based metric for the exact-match protocol. The reciprocal rank of a query is the multiplicative inverse of the rank of the first correct recommended team. The mean reciprocal rank is then the average of the reciprocal ranks of the results of the query set Q. 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:

$$MRR = \frac{1}{|Q|} \sum_{p \in Q} \frac{1}{rank_p} \tag{23}$$

If the set of recommended teams does not contain a match, then $\frac{1}{rank_P}$ is defined to be 0.

4.7.3 Mean Rank (MR) and Mean Rank of Hits (Hit MR)

MR quantifies the average rankings of the first matched teams of the query set Q. If the ranking of the first matched team is larger than 100, its rank is set to 101 to avoid including heavy outliers into the computation. Hit MR is similar to MR, but only considers recommended lists that contain a matched team. Note that, since the Max-Logit algorithm does not enumerate all the possible combinations of the teams, there may be cases where the algorithm terminates (i.e. reaches the maximum number of iterations) without discovering a matched team.

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

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

5 Experiment, Results, and Discussion

This section presents the datasets used for model validation, evaluation protocols, experimental results, and related discussion. The evaluation is conducted in two parts. The first part is to identify the most suitable classification algorithm for the team-fitness scoring function. The latter validates the efficacy of the proposed software team recommendation mechanism. We focus on the following research questions:

- **RQ1: Is it possible to quantify the ability of a software team to successfully resolve a given software development task?** Every ranking-based recommendation algorithm relies on the ability to quantify the relevance of items, so they can be ranked and recommended (Zhang et al. 2020b). In the context of our research, such items are software team configurations. Hence, the ability to accurately quantify such a relevance score (i.e. the team-fitness score) for a given software team, that allows more suitable teams to be ranked higher than less worthy ones, is crucial for the further development of the software team recommendation algorithm.
- **RQ2: Is it possible to recommend software teams that comply with the role requirements and are suitable for a given software development task?** Once RQ1 is satisfactorily answered, the next question would be how to integrate the team-fitness scoring function with a recommendation framework, that takes the role requirements and the task description from the task assignee, and suggests a ranked list of top software teams for the task.
- **RQ3: Can the proposed software team recommendation method be adopted for single-role recommendation tasks?**: A natural question would arise as to whether the proposed software team recommendation algorithm could be modified such that it recommends members only for particular roles, instead of the whole team. For example, the task assignee may want to specifically choose a particular member for the Developer role, and let the recommender suggests suitable members for the Reviewer role. This research question explores the possibility of doing so.

5.1 Datasets

Three real-world software development datasets from three open-source well-known software systems (i.e. Moodle, Apache, and Atlassian) hosted in the Jira platform are used to validate our method. Table 3 summarizes the statistics of these datasets in terms of task sizes, components, roles, and data collection periods. The definition of a successful task is defined in Section 4.2.2. Note that, Atlassian has a significantly less proportion of successful tasks (1.2%). This is because most of Atlassian tasks do not clearly specify the role of each member. Hence, these are filtered out due to incompatibility with our assumption that a

Table 3 Statistics of the selected datasets

Statistics \Datasets	Moodle	Apache	Atlassian
# Issues	88,655	507,319	238,322
# Successful Issues	27,284	43,196	2,917
# Components	315	782	170
# Developers	450	2,265	39
# Testers	195	–	21
# Reviewers	133	41	127
# Integrators	16	–	–
# Direct Messages	21,268 (4.18%)	339,694 (14.2%)	36,441 (6.14%)
# Non-direct Messages	487,599	2,053,346	556,893
Period	2002/09/05 - 2019/05/22	2002/04/03 - 2019/07/23	2004/11/20 - 2019/03/29

team member must have a role. On the other hand, most Moodle issues explicitly label each member with a role, and therefore has the largest proportion of successful teams, compared to the other two datasets. The successful issues of each dataset are separated chronologically into 80% training and 20% testing sets. The training/testing data is split chronologically to avoid the model to learn from future issues. The 80% training data is used to generate the knowledge graphs and evaluate the team-fitness scoring functions using 5-fold cross validation. The other 20% of the data is used to validate the team recommendation methods.

In the Moodle dataset, besides the assignee, the reviewer, tester, and integrator roles are explicitly labeled. In the Apache dataset, only the reviewer roles are explicitly given. In the Atlassian dataset, the developer, reviewer, and tester roles are explicitly annotated. Note that the developer roles in some systems such as Moodle and Apache are not explicitly labeled. To identify these developers, we make an assumption that a team member who has committed the code to the corresponding GitHub repository is deemed as a developer. The GitHub auto-generated commit logs can be easily extracted and parsed from a Jira issue.

Figure 7 illustrates the distribution of successful and non-successful tasks of the three datasets over time. Note that, while there are more non-successful tasks than the successful ones at a given time, it does not mean that these non-successful tasks are failed ones - they simply do not match the definition of a successful task defined in Section 4.2.2. Furthermore, the numbers of successful tasks of Apache and Atlassian datasets start to emerge quite late compared to the non-successful ones. This is because, before these periods, each task

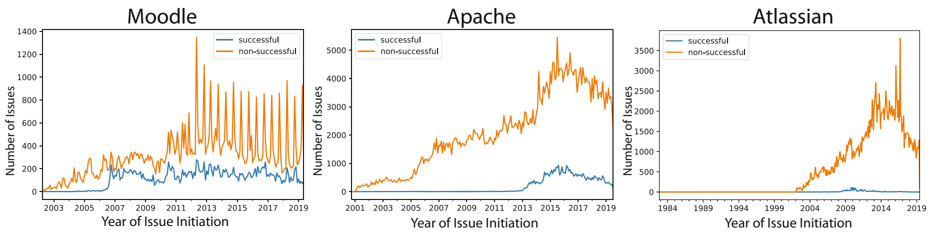


Fig. 7 Distribution of successful and non-successful tasks in terms of task initiation time (year)

could be either small or did not have sufficient information to identify necessary members' roles, hence is treated as a non-successful task.

Figure 8 illustrates the distributions of successful and non-successful teams in terms of the team size (number of members including the task assignee) of the three datasets. A task with zero team size means that it does not have a designated task assignee and members' roles cannot be identified.

5.2 Evaluation of Scoring Function

This section report the empirical results of the proposed team-fitness scoring function that answer RQ1: Is it possible to quantify the ability of a software team to successfully resolve a given software development task?

The probability output by a binary classifier, which predicts whether a candidate team is effective or not, is used as the team-fitness score. The score ranges in [0,1] where 1/0 indicates that the candidate team is perfectly suitable/unsuitable for the given task description and task assignee. Different machine learning classification algorithms are evaluated based on their ability to fit the training data, where positive samples are generated from actual successful teams, and the negative samples are synthesized by randomizing team members.

5-fold stratified cross-validation is performed on the training data. The area under the receiver operating characteristic curve (i.e. ROC-AUC) is used as the main evaluation metric, due to its ability to quantify the model fitness to the data, regardless of the cut-off probability. Furthermore, traditional classification metrics such as precision, recall, F1 of the positive class (successful teams), and MCC (Matthews Correlation Coefficient) are also reported. The ROC-AUC value of 0.5 is equivalent to random guess, while the value of 1.0 indicates a perfect fit.

Table 4 summarizes the average ROC-AUC, precision, recall, F1, and MCC values of each classification model on the three datasets. In terms of precision, RF is the top performer for the Moodle and Apache datasets, while ANN yields the highest precision for the Atlassian dataset. It is worth noting that some classifiers have relatively poor precision in general, namely LR, NB, and QDA. We suspect that this may be caused by the class imbalance issues in the training data that bias the models towards the majority class (i.e. non-successful). In terms of recall, LR gives the highest recall for the Moodle and Atlassian datasets, while DT for the Apache dataset.

F1 represents the single metric that reflects both precision and recall. We found that RF yields the highest F1 for both the Moodle and Atlassian datasets with F1 of 0.890 and 0.460 respectively, while DT for Apache with F1 of 0.991. Table 5 summarizes the p-values from the Mann-Whitney U Test between RF and other classification algorithms in terms of ROC, precision, recall, and F1. While DT yields the top F1 for the Apache dataset

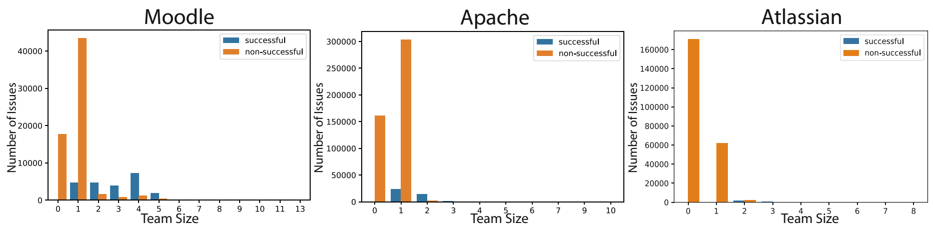


Fig. 8 Distribution of successful and non-successful tasks in terms of team size (number of members)

Table 4 Comparison of areas under the receiver operating characteristic curves (ROC), precision, recall, F1, and MCC of different classification algorithms

Algorithm	ROC			Precision			Recall			F1			MCC		
	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian
LR	0.987	0.979	0.954	0.260	0.253	0.167	**0.956**	0.934	**0.863**	0.409	0.399	0.280	0.703	0.740	0.427
DT	0.942	0.993	0.778	0.812	0.984	0.338	0.896	**0.998**	0.537	0.852	**0.991**	0.415	0.873	0.977	0.488
RF	**0.993**	**0.994**	**0.981**	**0.955**	**0.987**	0.656	0.833	0.984	0.354	**0.890**	0.986	**0.460**	**0.912**	**0.984**	**0.543**
NB	0.978	0.962	0.931	0.187	0.283	0.146	0.860	0.827	0.777	0.308	0.421	0.246	0.408	0.517	0.335
QDA	0.983	0.966	0.928	0.223	0.307	0.144	0.949	0.786	0.793	0.361	0.441	0.243	0.479	0.541	0.332
kNN	0.952	0.918	0.808	0.915	0.893	0.617	0.827	0.743	0.334	0.869	0.811	0.433	0.822	0.922	0.504
ANN	0.989	0.894	0.978	0.906	0.975	**0.661**	0.852	0.935	0.399	0.878	0.954	0.482	0.823	0.830	0.471

Table 5 Comparison of p-values from the Mann-Whitney U Test between Random Forest and other classification algorithms

Algorithm	ROC			Precision			Recall			F1		
	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian	Moodle	Apache	Atlassian
LR	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	0.07
DT	<0.05	<0.05	<0.05	<0.05	0.27	<0.05	<0.05	0.09	<0.05	0.11	0.34	0.5
NB	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	0.42	<0.05	<0.05	<0.05	<0.05	0.07
QDA	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	0.07
kNN	<0.05	<0.05	<0.05	<0.05	<0.05	0.27	0.5	<0.05	0.5	0.20	<0.05	0.50
ANN	<0.05	<0.05	<0.05	0.34	0.20	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05

(i.e. F1 = 0.991), it is only higher than that of RF (i.e. F1 = 0.986) negligibly by 0.5%. The statistical tests in Table 5 also confirm that the F1 values of RF and DT are not significantly different since the corresponding p-values are greater than 0.05. Hence, it can be inferred without loss of generality that RF is the best classifier for all the datasets in terms of F1. It is worth noting that while DT, RF, kNN, and ANN yield reasonably high F1 (i.e. > 0.8 for Moodle and Apache, and > 0.4 for Atlassian), the other classifiers have noticeably poor F1. Most of these poor-F1 classifiers give high recall but very low precision. Such a phenomenon may be caused by the imbalance of the datasets that leads to a bias towards the majority class, which could be mitigated using data balancing techniques such as SMOTE and under-sampling of the majority classes (Tantithamthavorn et al. 2018). However, from a preliminary investigation, we found that such data balancing techniques, while marginally improving the performance of some classifiers, did not improve the best classification results yielded by RF. Table 6 compares the classification results using different data-balancing methods, including SMOTE, under-sampling of the majority class, and over-sampling of the minority class, using RF as the classifier and 5-fold cross-validation protocol. According to the results, not employing any data-balancing method on the training data (i.e. None) yields the highest performance in almost all the metrics, except for recall where the under-sampling method gives a better recall value. However, careful investigation of the impact of data imbalance is left for potential future work without affecting the research objectives of this paper.

The area under the ROC curve (i.e. ROC) measures the overall fitness of the classification model to the training data regardless of the cut-off probability. While precision, recall, and F1 have been used as the main criteria for selecting classifiers, in our proposed method, we do not utilize the final classification results (i.e. effective vs. non-effective teams), but rather use the raw probability, which is the primary output of the classification algorithm, for ranking purposes. In another word, changing the cut-off probability affects precision, recall, and F1, but not the ROC. Therefore, ROC is used as the main evaluation metric to select the classifier for the team-fitness scoring function. Note that, ROC has also been used as the main criteria for many classification tasks to measure the overall discriminative power of the model rather than its final classification results (Hassan et al. 2017; Tuarob et al. 2020). Furthermore, if the model is used for ranking purposes, even with the presence of imbalanced data, the ordering of what the model perceives as positive cases is preserved and is unaffected by the different cut-off probability values (Jeni et al. 2013). From the experimental results in Table 4, RF gives the best ROC for all three datasets with ROC values of 0.993, 0.994, and 0.981 for Moodle, Apache, and Atlassian respectively. It is worth noting that, ROC values appear to be high even for generally ineffective classifiers such as LR, NB, and QDA. This is expected due to the highly imbalanced nature of the datasets.

Table 6 Comparison of classification performance yielded by different data balancing methods using Random Forest as the classifier

Metric	None	SMOTE	Under-Sampling	Over-Sampling
ROC	**0.993**	0.966	0.991	0.947
Precision	**0.955**	0.853	0.448	0.927
Recall	0.833	0.866	**0.926**	0.827
F1	**0.890**	0.859	0.604	0.874
MCC	**0.912**	0.892	0.663	0.910

Hence, other classification metrics such as precision, recall, and F1 are also used as the second criteria for the model selection.

From the above experimental results, it is evidenced that RF yields the highest ROC for all the datasets. Furthermore, in terms of classification performance, RF also achieves relatively high F1 compared with other classifiers even with the presence of imbalanced data. These could be due to the following reasons:

- RF is built with a multitude of decision trees, which is suitable for ruled extracted features, each of which indicates a soft signal that represents the class attribute (Breiman 2001). Since all of our features are computed using a set of rules and mathematical expressions, rather than using natural representations such as term-weights (for documents) or RBG values (for images), it is not surprising that tree-based classifiers such as RF and DT would be suitable for the features characterizing our datasets.
- RF has the capability to avoid over-fitting issues due to the randomness mechanism (Yi et al. 2019). This allows RF to achieve better recall while maintaining high precision.
- RF has a built-in automatic feature-selection mechanism (Zhang et al. 2020a). Specifically, randomly selecting a subset of features to construct a decision tree enables the algorithm to recognize important features, resulting in overall high classification performance.
- RF has been reported to have the ability to tolerate class imbalance without having to rely on data balancing techniques (Khoshgoftaar et al. 2007). Such capability is especially crucial to handle our imbalanced datasets.

Such analysis also aligns with the previous studies that found RF to be suitable classifiers for the tasks where features are rule-extracted and datasets are imbalanced (Tuarob et al. 2018; Tuarob et al. 2020). Therefore, RF is used to generate the team-fitness scoring functions for the subsequent team recommendation evaluation.

It is also worth noting that classification performance for the Atlassian dataset is much lower than that of the other two datasets in terms of precision, recall, and F1. This could be due to the insufficient training data of the Atlassian dataset with only 2,917 positive samples (due to excessive removal of *incomplete* software teams), compared to Moodle and Apache with 27,284 and 43,196 positive samples respectively. Therefore, as part of possible future work, we could harvest more samples from such a software system, or investigate semi-supervised techniques to make use of unlabelled samples (Zhang et al. 2017).

5.3 Evaluation of Team Recommendation

This section explains the evaluation protocol and discusses the empirical results of the proposed software team recommendation algorithm, which answer RQ2: Is it possible to recommend software teams that comply with the role requirements and are suitable for a given software development task?

In the software team recommendation task, *RECAST* takes a task description, task assignee, and required team roles as input, and recommends a ranked list of K teams based on the team-fitness scores. To evaluate the efficacy of this task, standard evaluation protocols for recommender systems are employed. Specifically, the method is validated based on its ability to guess the team configurations of the future (successful) tasks in the testing set. For each dataset (See Section 5.1), 20% most recent tasks are allocated for testing, while the other 80% is used for generating required knowledge networks as discussed in Section 4.3 and training the scoring function (See Section 4.5).

5.3.1 Baseline Algorithms

To the best of our knowledge, we are not aware of any previously proposed team recommendation algorithms that are specifically designed for collaborative software development. Hence, in terms of recommendation performance, we compare our proposed algorithm with a state-of-the-art team recommendation algorithm proposed by Liu et al. (2014) (i.e. Liu), and a baseline that randomly selects role-compatible users to fill in each required role (i.e. Random).

The method proposed by Liu et al. (2014) utilizes the Max-Logit algorithm to enumerate potential team combinations, and a machine learning classifier (i.e. Logistic Regression) as the scoring function. However, their scoring function is trained with the features defined in Section 4.4.1. Hence, the Liu method recommends teams based primarily on collaboration history and overall experience of the team members, without considering the task requirements, skills, and sentiment aspects. Our proposed algorithm, *RECAST*, extends such a previous method by also incorporating the task requirements and task assignee into the prior criteria so that the recommended teams would be crafted to also be competent with the task requirements and compatible with the task assignee. The scoring function of *RECAST* is also trained with a set of heterogeneous features that reflect not only the collaboration history and task experience, but also the team technical skills and social compatibility, characterizing effective and practical software development teams.

5.3.2 Software Team Recommendation Results

This subsection reports the recommendation results of *RECAST* against the two baselines on the three datasets. The evaluation is conducted using both the exact-match and the half-match protocols.

Table 7 summarizes the recommendation evaluation results using the exact-match protocol. Such a protocol considers a team as a match if the members of that team are exactly the same as those of the ground-truth team. It is apparent that *RECAST* outperforms the two baselines in all aspects. In terms of MRR, *RECAST* outperforms the best baseline (Liu) by 743.21%, 557.12%, and 639.03% for Moodle, Apache, and Atlassian datasets respectively. The MR values show that, overall, all the three methods find the exact-matched teams within the top 100 recommendations, while *RECAST* still outperforms the other two. It is interesting to note that if the Max-Logit algorithm finds a matched team, it will be ranked in the top three results for Moodle and Apache datasets, and top 20 for Alassian, compared to the best baseline that finds the exact-matched teams in the ranks of 30th-40th (if a matched team is found). This means that *RECAST* can perform well when Max-Logit can discover a matched team before the algorithm reaches termination. Note that, solving the issue where Max-Logit cannot find matched teams is trivial. To do so, one could increase the maximum number of iterations, or simply exhaustively enumerating all possible teams (if the dataset is small).

In terms of precision, Precision@10 of *RECAST* is better than the best baseline by 593.42%, 365.71%, and 305% for Moodle, Apache, and Atlassian, respectively. However, the relative precision differences are smaller for Precision@100 (i.e. 239.88%, 46.88%, and 2.42%). This is because the baseline correctly guesses the matched teams in the ranks 10th or higher on average, while *RECAST* performs better at guessing the correct teams within the top 10 results. Figure 9 displays precision at each number of recommendations (K) for all the three datasets, using the exact-match protocol. It is evident that *RECAST* can guess the correct teams in the top few results, compared to the other two baselines. The precision

Table 7 Exact-match evaluation results of different software team recommendation algorithms (Random, Liu, and RECAST) on datasets Moodle, Apache, and Atlassian

Exact Match	Moodle				Apache				Atlassian			
	Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)
MRR	0.002	0.010	**0.084**	743.21	0.004	0.010	**0.067**	557.12	0.062	0.041	**0.306**	639.03
MR	98.988	98.406	**88.76**	9.79	97.959	97.685	**93.773**	4.00	61.980	64.832	**47.230**	27.15
Hit MR	42.243	30.356	**2.991**	90.15	40.437	34.193	**1.851**	94.59	40.302	43.831	**18.016**	58.90
Precision@10	0.002	0.017	**0.119**	593.42	0.012	0.015	**0.072**	365.71	0.117	0.102	**0.413**	305.00
Precision@100	0.034	0.037	**0.125**	239.88	0.050	0.050	**0.073**	46.88	0.643	0.633	**0.648**	2.42
MAP	0.000	0.009	**0.084**	786.07	0.003	0.009	**0.067**	626.80	0.046	0.026	**0.299**	1068.73

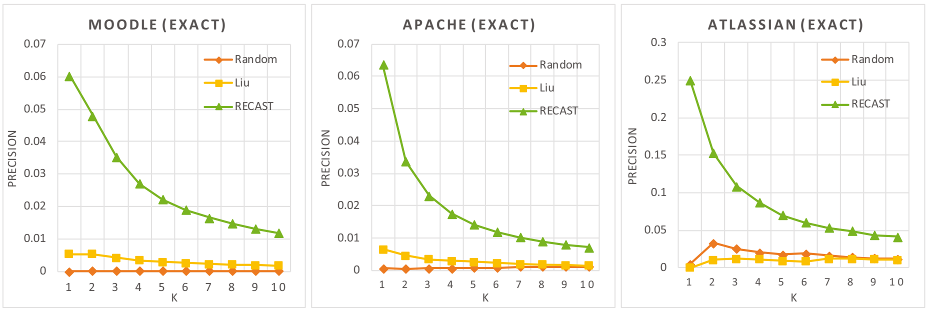


Fig. 9 Precision@K plots of different software team recommendation algorithms (Random, Liu, and RECAST) on datasets Moodle, Apache, and Atlassian, using the *exact-match* evaluation protocol

becomes decreasing as K increases because more non-matched teams are added to the top K results. It is worth noting that, the precision of the Random method is better than the Liu method for dataset Atlassian especially in the first 10 values of K. This is because the Atlassian dataset is much smaller compared to the other two datasets with only a small number of candidate members for each role (i.e. 39 developers, 21 testers, and 127 reviewers). Hence, it would be relatively easy for the Random method to randomly guess the correct answers, compared to the Liu method.

The recommendation results using the half-match protocol are summarized in Table 8. A recommended team is a half-match if at least half of the team members overlap with the ground-truth team. Loosely speaking, a half-matched team is said to be *similar* to the correct team. This evaluation protocol allows some flexibility in the evaluation, while still reflects the quality of the recommended results. Hence, it is observed that the overall half-match performance is better than that of the exact-match evaluation, regardless of the recommendation methods and datasets, except for Hit MR. This is because, with the half-match protocol, an enumerated team has a higher chance to be a match. Since Hit MR only considers the recommendation lists that contain a match, it is the case that more lists that contain half-matched results in the lower ranks additionally contribute to the values, and as a result, lower the overall Hit MR. This phenomenon also holds for the Liu method which also uses Max-Logit to enumerate teams.

Figure 10 illustrates the precision at each number of recommended teams (K) of the three algorithms on the three datasets. The analysis is similar to the exact-match evaluation. It is worth noting that, the precision of *RECAST* on some datasets (i.e. Moodle and Atlassian) remains high and does not decline as quickly, compared to the exact-match analysis in Fig. 9. This is because *RECAST* can find and recommend different variants of teams similar to the correct team in the top results.

From both the exact-match and half-match evaluation results, it is evident that the proposed *RECAST* algorithm can produce more accurate recommended teams than the two baselines. Statistical analyses using Mann-Whitney U Test ($\alpha = 0.05$), shown in Table 9, confirm that the performance in terms of MRR of *RECAST* is significantly different from that of the baselines in both the exact-match and half-match evaluation protocols. In terms of implementation, since an elbow points are observed around $K = 5$ from Figs. 9 and 10, it is our suggestion that the system produces only five recommended teams to prevent the choice overloading issue, while still maintaining the quality of the recommendation.

Table 8 Half-match evaluation results of different software team recommendation algorithms (Random, Liu, and RECAST) on datasets Moodle, Apache, and Atlassian

Half Match	Moodle				Apache				Atlassian			
	Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)
MRR	0.011	0.019	**0.175**	799.99	0.004	0.011	**0.072**	569.86	0.102	0.070	**0.403**	473.60
MR	94.361	93.645	**74.383**	20.57	97.631	97.331	**93.083**	4.36	48.541	50.133	**30.765**	38.63
Hit MR	32.938	30.329	**6.930**	77.15	41.427	34.734	**3.953**	88.62	37.137	39.457	**16.025**	59.39
Precision@10	0.039	0.051	**0.920**	1698.68	0.014	0.018	**0.109**	490.40	0.270	0.199	**1.082**	443.59
Precision@100	0.394	0.348	**3.352**	863.80	0.066	0.070	**0.180**	157.35	2.117	1.908	**2.296**	20.32
MAP	0.007	0.017	**0.159**	837.45	0.003	0.010	**0.071**	646.35	0.078	0.050	**0.386**	675.38

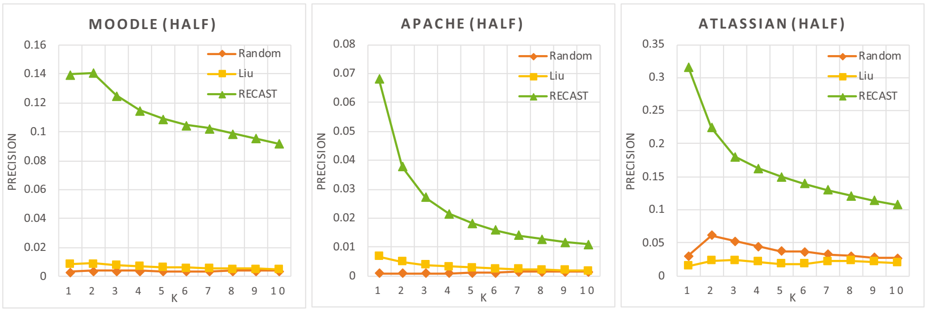


Fig. 10 Precision@K plots of different software team recommendation algorithms (Random, Liu, and RECAST) on datasets Moodle, Apache, and Atlassian, using the *half-match* evaluation protocol

Figure 11 shows examples of recommended teams for a software task by our proposed *RECAST* and the two baseline algorithms using a real-world task description and assignee. The system name and all usernames are anonymized. This task (i.e. issue) is a bug-fix task where the error stems from the miscommunication between the main system *ABC* and the plugin application *flashcards*. The *ABC* system is supposed to not convert the primitive type of the data files (i.e. *fdk*) to a *zip* format that prevents *flashcards* from directly opening them. The task assignee (i.e. *U100*) requires a team of three members to work on this issue, including an integrator, a tester, and a reviewer. The actual assignments of these roles are *U101*, *U102*, and *U103* respectively.

The three software team recommendation algorithms, namely Random, Liu, and our *RECAST*, are used to suggest appropriate 10 teams for this task. The teams with $*$ and $+$ symbols are those that are exact-matched and half-matched respectively with the actual team. The ***bold-italic*** usernames are those that are also in the actual team (regardless of the roles). It is evident that from this particular example, *RECAST* is able to recognize the actual team in the first rank, and six more similar (half-matched) teams within the top 10 results. The Liu method is not able to capture any exact-matched or half-matched teams in the top 10 ranked results. Furthermore, the Random method cannot guess any correct team members in the first 10 ranked results.

5.4 Single-Role Recommendation

This section reports the adaptation of *RECAST* to recommend members for a particular role instead of the whole team with various role requirements. The empirical results also fortify the answers for RQ3: Can the proposed software team recommendation method be adopted for single-role recommendation tasks?

Table 9 Statistical tests using Mann-Whitney U Test on MRR between RECAST and the other baselines for the team recommendation task

Evaluation Protocol	Moodle		Apache		Atlassian	
	Random	Liu	Random	Liu	Random	Liu
Exact-Match	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
Half-Match	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05

Issue Summary	Treat the fdk extension or avoid extension renaming for unknown types		
Issue Description	In our school, students use the app flashcards: It generates an fdk file. When I upload an fdk file in a course, and ask students to download it, the ABC app let it to be opened by the app flashcards, but a #er that, they get an error. Apparently, the file is converted to a zip file and thus, it cannot be recognized by the flashcards app. Can this error be fixed?		
Assignee	U100		
Actual Team	Integrator: U101, Tester: U102, Reviewer: U103		
Rank	**Random**	**Liu**	**RECAST**
1	U104, U105, U106	U137, U146, U147	***U101*** ,***U102*** ,***U103***
2	U104, U105, U107	***U101***, U146, U147	+ ***U101*** ,***U103***, U105
3	U108, U109, U106	***U101***, U148, U147	+ ***U101***, U104, ***U103***
4	U104, U110, U106	U113, U148, U147	+ ***U101***, U112, ***U103***
5	U104, U105, U111	U118, U148, U147	+ ***U101***, U161, ***U103***
6	U104, U105, U112	U118, U148, U149	U108, ***U103***, U105
7	U113, U105, U106	U150, U148, U149	U125, ***U103***, U105
8	U108, U105, U106	U150, U151, U149	***U101***, U137, U105
9	U104, U105, U114	U150, U152, U149	+ ***U101*** ,***U103***, U127
10	U104, U105, U115	U143, U152, U149	+ ***U101*** ,***U103***, U162

Fig. 11 Example recommended teams using the Random, Liu, and RECAST methods for a real-world software development task and assignee. Usernames are anonymized

While *RECAST* has shown to perform well in recommending effective software teams, compared to the state-of-the-art team recommendation algorithm, it is only natural that one might be curious if the algorithm could be applied to the situation where the task assignee only wants suggestions for a particular role (i.e. not the whole team). To investigate this possibility, we modify the Max-Logit algorithm to only enumerate possible members for a particular role, while using actual team members for the other roles. This modification to the Max-Logit algorithm allows us to compare *RECAST* with the state-of-the-art Liu method, due to using the same Max-Logit algorithm as the candidate team enumerator. We also compare the results with a randomization-based method that randomly selects role-compatible members to fill in the target role. The set of roles identified in the datasets include *Developer*, *Reviewer*, *Tester*, and *Integrator*. The exact-match evaluation protocol is used to validate the three algorithms.

Table 10 summarizes the single-role evaluation of the Random, Liu, and *RECAST* algorithms on each role of the three datasets. The table also reports the weighted average results. *RECAST* outperforms the state-of-the-art baseline (Liu) on all aspects, except for the Hit@100 of the Reviewer and Tester roles of the Moodle dataset. However, such performance differences in Hit@100 are only marginal (i.e. worse by 0.6% and 2.05% for the Reviewer and Tester roles respectively). *RECAST* performs exceptionally well for the Atlassian dataset with perfect evaluation performance on the Developer role. This may be because there are only a handful of developers in the Atlassian dataset (i.e. 39 users). Note that, not all the roles are available in all datasets; hence, the statistics for these unavailable roles (Tester and Integrator of Apache, and Integrator of Atlassian) are not reported. Statistical analyses using Mann-Whitney U Test ($\alpha = 0.05$), shown in Table 11, confirm that the performance in terms of MRR of *RECAST* is significantly different from that of the baselines.

Table 10 Single-role evaluation results of different software team recommendation algorithms (Random, Liu, and RECAST) on datasets Moodle, Apache, and Atlassian

Role	Metric	Moodle				Apache				Atlassian			
		Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)
Developer	MRR	0.02	0.08	**0.20**	160.49	0.00	0.01	**0.04**	371.30	0.08	0.10	**1.00**	898.88
	MR	80.43	76.53	**62.78**	17.96	98.68	98.34	**96.60**	1.77	13.76	21.59	**1.00**	95.37
	Hit MR	47.58	38.27	**6.15**	83.93	51.98	38.25	**3.55**	90.71	13.76	21.59	**1.00**	95.37
	Precision@10	0.04	0.12	**0.41**	229.48	0.00	0.01	**0.04**	225.84	0.28	0.21	**1.00**	383.33
	Precision@100	0.44	0.45	**0.46**	2.24	0.05	0.04	**0.05**	6.29	1.00	1.00	**1.00**	0.00
	MAP	0.01	0.07	**0.20**	184.41	0.00	0.01	**0.04**	419.10	0.03	0.07	**1.00**	1424.09
Reviewer	MRR	0.04	0.07	**0.21**	208.76	0.16	0.24	**0.41**	70.39	0.04	0.05	**0.33**	534.33
	MR	60.41	57.44	**36.39**	36.64	16.60	17.87	**5.73**	67.91	53.52	60.13	**36.39**	39.48
	Hit MR	47.60	42.90	**14.32**	66.63	16.60	17.87	**5.73**	67.91	43.95	49.28	**24.64**	49.99
	Precision@10	0.10	0.14	**0.43**	206.48	0.57	0.35	**0.87**	147.62	0.10	0.12	**0.50**	323.53
	Precision@100	0.76	0.75	**0.75**	-0.60	1.00	1.00	**1.00**	0.00	0.92	0.87	**0.92**	4.80
	MAP	0.02	0.05	**0.20**	281.45	0.14	0.21	**0.41**	90.44	0.02	0.04	**0.32**	787.33
Tester	MRR	0.03	0.05	**0.16**	191.25	No Tester Roles				0.12	0.20	**0.86**	332.67
	MR	75.34	70.62	**55.06**	22.03					10.96	10.40	**2.27**	78.19
	Hit MR	50.06	42.82	**11.18**	73.90					10.96	10.40	**2.27**	78.19
	Precision@10	0.07	0.12	**0.35**	192.27					0.56	0.50	**0.98**	96.15
	Precision@100	0.50	**0.52**	0.51	-2.05					1.00	1.00	**1.00**	0.00
	MAP	0.02	0.04	**0.15**	241.75					0.09	0.16	**0.85**	422.45

Table 10 (continued)

Role	Metric	Moodle				Apache				Atlassian			
		Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)	Random	Liu	RECAST	Diff (%)
Integrator	MRR	0.21	0.27	**0.54**	101.57	No Integrator Roles				No Integrator Roles			
	MR	8.61	7.62	**3.99**	47.61								
	Hit MR	8.61	7.62	**3.99**	47.61								
	Precision@10	0.61	0.68	**0.94**	38.49								
	Precision@100	1.00	1.00	**1.00**	0.00								
	MAP	0.18	0.24	**0.53**	119.82								
Average	MRR	0.07	0.12	**0.28**	137.65	0.00	0.01	**0.04**	312.06	0.06	0.09	**0.54**	485.20
	MR	55.65	52.52	**38.99**	25.76	97.95	97.63	**95.80**	1.88	38.50	43.60	**23.89**	45.21
	Hit MR	33.24	28.91	**8.66**	70.04	46.38	34.70	**3.91**	88.73	31.00	34.72	**15.49**	55.39
	Precision@10	0.21	0.27	**0.53**	99.30	0.01	0.02	**0.05**	210.91	0.23	0.22	**0.68**	210.20
	Precision@100	0.68	**0.69**	0.68	-0.23	0.06	0.05	**0.05**	5.20	0.95	0.92	**0.95**	2.91
	MAP	0.06	0.10	**0.27**	164.77	0.00	0.01	**0.04**	355.88	0.04	0.07	**0.53**	665.37

Table 11 Statistical tests using Mann-Whitney U Test on MRR between RECAST and the other baselines for the single-role recommendation task

Role	Moodle		Apache		Atlassian	
	Random	Liu	Random	Liu	Random	Liu
Developer	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
Reviewer	<0.05	<0.05	<0.05	<0.05	<0.05	<0.05
Tester	<0.05	<0.05	–	–	<0.05	<0.05
Integrator	<0.05	<0.05	–	–	–	–

It is also interesting to note that, the average single-role performance is overall better than the exact-match team recommendation performance reported in Table 7, with MRR of 28%, 4%, and 54% compared to exact-team MRR of 8.4%, 6.7%, and 30.6% for Moodle, Apache, and Atlassian datasets respectively. Note that recommending developers for Apache has a noticeably lower accuracy than the other datasets. This is because the Apache dataset has the largest pool of 2,265 developers. Hence, predicting the correct developers for Apache could be relatively difficult due to the larger candidate pool, compared to the other datasets. Regardless, *RECAST* remains the top performer with 371.30% improvement, in terms of MRR, over the best baseline on average.

Figure 12 plots average precision at each number of recommended candidates (K) of the three algorithms for datasets Moodle, Apache, and Atlassian. Overall, *RECAST* performs better than the other two baselines at every K. The prevalent difference is observed during the first few results, meaning that *RECAST* is able to correctly predict the correct team members at the very top-ranked results, compared to the Random and Liu methods.

5.5 Parameter Sensitivity

While *RECAST* has shown to be effective at recommending software teams with promising results compared to the baselines, this section discusses how varying certain parameters could affect the recommendation performance.

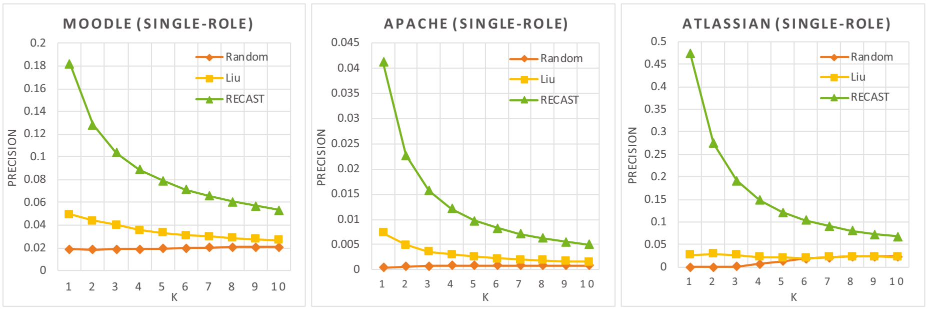


Fig. 12 Average Precision@K plots of different single-role recommendation algorithms (Random, Liu, and RECAST) on datasets Moodle, Apache, and Atlassian

5.5.1 Optimal Numbers of Topics

The similarity between two tasks in the task similarity network (Section 4.3) is quantified using topical similarity between the topic distribution vectors of the two tasks. Each task's topic distribution is derived using the Latent Dirichlet Allocation (LDA) algorithm that generates a topical model for each dataset. Here, we try both the regular (non-labeled) LDA (Blei et al. 2003) and labeled LDA (Ramage et al. 2009) using issue components as prior labels.

One of the hyperparameters that govern the creation of topical models is the number of topics. Too few topics would result in general topics that do not distinguish documents well; while too many topics can lead to random, meaningless topics that comprise idiosyncratic word combinations (Steyvers and Griffiths 2007). Hence, choosing a suitable number of topics is key to achieve optimal topic models.

Multiple studies that utilized topic modeling in various applications have shown that the effectiveness of the learned topics correlates with the coherence of the term distribution in each topic (Deerwester et al. 1990; Hofmann 2001; Tuarob et al. 2015). Newman et al. defined the coherence of a topic as the ability to be interpreted as a meaningful topic by examining the term composition (Newman et al. 2010). They also proposed a set of methods for automatic evaluation of topic coherence, namely WordNet, Wikipedia, and Google based methods. In our experiment, we adopt a similar scheme as their Wikipedia-based method, using pair-wise mutual information (PMI), since this scheme was reported the most accurate in their work.

Since we aim to find the optimal number of topics that results in the highest topic coherence, the average topic coherence score is calculated for each topic set. Mathematically, let $Z_T = \{z_1, z_2, ..., z_T\}$ be the set of T learned topics. We aim to calculate the average topic coherence (ATC) score for the topic set Z_T by taking the arithmetic mean of the coherence scores of all the topics in Z_T, as follows:

$$ATC\,(Z_T) = \frac{1}{|T|}\sum_{z \in Z_T} C(z) \tag{26}$$

$C(z)$ is the coherence score of the topic z, and is calculated as follows:

Let W_{10} be the set of top 10 words in z. $C(z)$ is then the average of pair-wise mutual information (PMI) scores of all possible unique pairs of the words in $W_{10} = \{w_1, w_2, ..., w_{10}\}$.

$$C(z) = \frac{1}{45}\sum_{i,j \in \{1,...,10\}, i<j} PMI(w_i, w_j) \tag{27}$$

$$PMI(w_i, w_j) = \log\left(\frac{p(w_i, w_j)}{p(w_i)\cdot(w_j)}\right) \tag{28}$$

Where $p(w_i)$ and $p(w_j)$ are the probability of w_i and w_j, estimated by the proportion of the documents that contain at least one occurrence of w_i and w_j respectively. Similarly, $p(w_i, w_j)$ is the proportion of the documents that contain both w_i and w_j.

Instead of using Wikipedia articles as the external knowledge source as used by Newman et al. (2010), we use the task descriptions in our datasets as the external knowledge. This is because most of the task descriptions in our datasets are from very specific domains that concern only software specification and requirement, which Wikipedia does not well cover. Furthermore, these software task descriptions contain a number of technical terms, software keywords, and component names, which are not normally used in general encyclopedias.

For each dataset, we use all the task descriptions in the training set to model topics. The numbers of topics $T \in \{100, 200, 300, 400, 500\}$ are used. At each T, the non-labeled LDA and labeled LDA are run with 3,000 iterations to learn a set of T topics, Z_T. Then, the average topic coherence scores are computed for each Z_T.

Figure 13 plots the average topic coherence (ATC) from the non-labeled LDA and labeled LDA of the three datasets, where a higher value of ATC represents a better quality of the topic set. For Moodle, the peak ATC is achieved using non-labeled LDA with 300 topics. The peak of ATC for the Apache dataset is observed at $T = 400$ using labeled LDA. Finally, for the Atlassian dataset, the highest ATC is spotted when $T = 200$ using labeled LDA. Hence, these configurations are used in *RECAST* to build topic models for task similarity calculation as discussed in Section 4.3.

It is worth noting that, labeled LDA tends to perform better than its non-labeled version due to being supervised and able to incorporate additional knowledge from labels (i.e. issue components). This is the case for Apache and Atlasstian datasets. However, it is observed that the non-labeled LDA achieves higher ATC at every T for the Moodle dataset. After investigating this phenomenon carefully, we find that Moodle has relatively more spurious components, with each component representing specific features in the Moodle system, compared to the other two datasets. This conjecture is also supported by the average number of tasks per component: 218 (Moodle), 649 (Apache), 1,402 (Atlassian). These numbers represent the average number of documents for each class when training a topic model. It follows that Moodle has the lowest documents/class which could lead the labeled LDA to ineffectively learn the topics compared to allowing the non-labeled LDA to automatically figure out topics on its own.

5.5.2 Feature Analysis

The team-fitness scoring function is trained with 20 features, a majority of which are extracted from the knowledge networks discussed in Section 4.3. These features have different levels of impact on the scoring function's ability to assign an accurate score to a team. This section investigates the importance of these features using the feature importance scores output as by-products from the Random Forest model.

Summarized in Table 12 are the lists of features used to train the team-fitness scoring functions along with their Random Forest's feature importance scores for the Moodle, Apache, and Atlassian datasets. The ***bold-italic*** figures are the top five features. Different

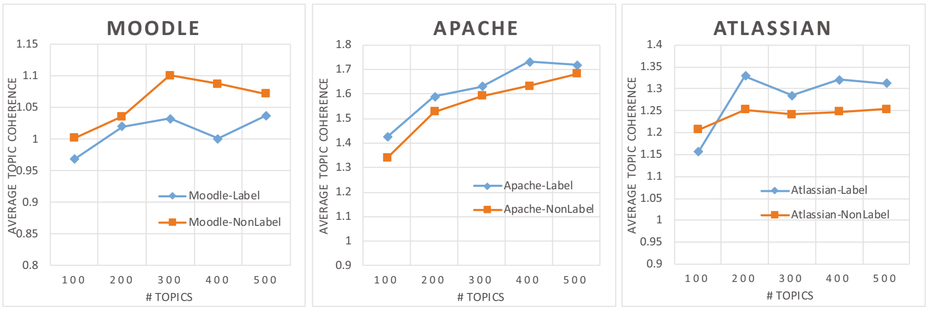


Fig. 13 Comparison of PMI values from various numbers of topics using both the labeled LDA and non-labeled LDA on the three datasets

Table 12 Feature importance scores calculated while training the scoring functions using Random Forest on datasets Moodle, Apache, and Atlassian

Feature	Moodle	Apache	Atlassian
Task Familiarity	**0.1539**	0.0147	0.0272
Task Proximity	**0.1175**	0.0472	**0.1245**
Relatedness to Assignee	0.0673	0.0458	0.0486
Team Coherence	**0.1621**	0.0524	**0.0527**
Team Positivity	0.0126	0.0036	0.0003
Team Criticism	0.017	0.0046	0.0003
Positivity towards Assignee	0.002	0.002	0.0001
Criticism towards Assignee	0.0031	0.0033	0.0001
Skill Competency	0.0028	**0.0799**	0.0047
Skill Diversity	0.0074	0.0064	0.0103
Component Experience	0.0345	**0.1315**	0.0078
Team Contribution	0.0192	0.0022	0.0227
Domain Experience	0.0315	**0.2229**	0.0516
Co-Task Frequency	0.0386	**0.1091**	**0.2877**
Experience	0.0355	0.062	0.0435
Role Experience	**0.0916**	0.0657	**0.0852**
Closeness	**0.1678**	**0.0921**	**0.1759**
Success Experience	0.0312	0.0526	0.0468
Connection	0.0017	0.0015	0.0004
Success Rate	0.0028	0.0005	0.0099

datasets have different natures and collaborative cultures as evident by different sets of top features across the three datasets.

In Moodle, *Closeness*, *Team Coherence*, *Task Familiarity*, *Task Proximity*, and *Role Experience* contribute as the top five features. Two of these features are related to experience with similar tasks, two related to collaboration experience, and one related to experience working on the required roles. It may be inferred that those task assignees in Moodle typically select team members based primarily on personal familiarity with the chosen members. It also appears that a software practitioner in Moodle tends to remain in the same roles across several tasks.

In Apache, *Domain Experience*, *Component Experience*, *Co-Task Frequency*, *Closeness*, and *Skill Competency* are ranked the top five features. Three of these features are indicators of experience with relevant tasks, while the other two features characterize collaboration history and required skills. Hence, it could be interpreted that task assignees in Apache tend to select members with relevant task experience that also have the necessary skills as required by the tasks.

In Atlassian, *Co-Task Frequency*, *Closeness*, *Task Proximity*, *Role Experience*, and *Team Coherence* are the top five features. For this dataset, there is no single consensus towards a single criterion to select team members. The top features represent the history of getting involved in past Atlassian tasks, collaboration history, current relevant tasks, and experience working in the roles. The *Co-Task Frequency* feature, representing the task experience with other team members, is prevalently important with the importance score of 0.2877, 475% higher than the average feature importance score. This makes sense since the Atlassian dataset only contains a handful of software practitioners. Hence, it may become obvious for a task assignee in Atlassian to pick a member based on their history of collaboration in the same tasks.

While the above feature analysis could shed light on common important features shared across the three datasets, it would be useful to also explore the influence of correlation/collinearity of features used to train the team-fitness scoring function (Jiarpakdee et al. 2019), which we plan to investigate as part of the potential future work.

5.6 Implications

We propose a set of novel features drawn from multiple knowledge graphs, constructed from real-world software systems hosted by Jira, that characterize not only the team's collective technical skills and collaboration history, but also their compatibility when members are working together. While every single feature serves as a soft signal that is not intended for absolute interpretation of a team's effectiveness, all the features in combination have shown quite effective when used to train machine learning classifiers (i.e. Random Forest) to recognize teams suitable for a given software development task (See Section 5.2). While previous studies have explored features to represent teams, they only reflect team skills and collaboration history without taking social aspects of teamwork into account. In this research, we showed that social aspects such as team coherence also contribute as important features for quantifying the effectiveness of software teams (See Section 5.5.2). We propose a novel algorithm to recommend software team configurations for a given software task. The algorithm utilizes Max-Logit to effectively prune the search space to avoid exhaustive enumeration of all the possible team combinations, and ranks the candidate teams using the team-fitness scoring function trained using a Random Forest classifier with the proposed features. To the best of our knowledge, we are the first to explore such team recommendation problems in the software engineering domain. The experimental results shown in Sections 5.3 and 5.4 illustrate that the proposed method outperforms the baselines by large margins in both team recommendation and single-role recommendation tasks. The minimal assumptions of the team and task structure allow the proposed methodology to generalize across multiple software platforms beyond Jira. The implications of this research are as follows.

5.6.1 Implications for Open-Source Software Projects

Open-source software (OSS) projects are typically contributed by a wide variety of developers with diverse skills, experience, and commitment. Indeed, studies have shown that causes of open-source project failure could stem from ineffective development teams that are technological limited and have conflicts among team members (Coelho and Valente 2017). Our proposed method incorporates features that represent both necessary technical skills and teamwork compatibility, which have been shown to contribute to project success (See Section 5.5.2). Therefore, applying the findings in real-world OSS projects could help to mitigate issues arising from ineffective team members and could result in an overall better project success rate.

5.6.2 Implications for Software Developers

The ability to automatically identify suitable team members to work on a software project could advocate productive collaboration in the software development ecosystem. For example, task assignees could easily find suitable replacements for team members who cease to contribute to the projects (Iaffaldano et al. 2019) in a timely manner, without significant delay or suspense on the progress. Furthermore, the proposed method can identify team members that contribute to the higher chance of task success, which could provide an alternative quantitative measure to reflect developers' performance and contribution, and help them to improve on the aspects that they lack.

5.6.3 Implications for Researchers

This work is part of a bigger picture of research that aims to study the computational aspects of collaboration dynamics in software projects. While we present the findings at the task level, this research could be building blocks for larger analyses in the project or ecosystem levels. Researchers could also extend this work in multiple directions such as recognizing influential developers or experts in OSS projects and identifying new roles and skills. Furthermore, most of the features introduced in Section 4.4 are handcrafted by previous researchers and us, and potentially are subject to controversy. Therefore, techniques that are less dependent on handcrafted features such as deep learning ones could be explored to mitigate the limitation of this work.

6 Threads to Validity

We identify the following threads to validity.

- **Platform Selection:** The case studies used to validate our proposed models were scraped from the Jira platform, whose main purpose is to facilitate software progress tracking. The platform is chosen due to the rich information of software team structures, task requirements, developers' skills, and interaction among team members. However, such a platform does not cover the actual development activities such as source code revision management and social interaction while writing code. While this limited information available in the Jira platform is sufficient for the proposed model to make a meaningful prediction, the model could be improved if information during the actual development can be incorporated.
- **Definition of Successful Tasks:** In this research, we define successful tasks as tasks that are resolved without reopening, and that the teams of such tasks are deemed good-enough teams. *RECAST* is trained to recommend teams that are similar to these good-enough teams, so to prevent task assignees from forming ineffective ones. However, while this definition of successful tasks works well in general, tasks that are not (yet) resolved or have been reopened are not necessarily unsuccessful. For example, a task may be on-going (hence, not yet resolved) or reopened because the task assignee does not want to create a new, similar task. However, without a clear definition of and a good methodology to determine *unsuccessful* tasks, we had to disregard these non-successful tasks, which account for the majority of the tasks in our datasets, resulting in a major loss of valuable information from which the model could learn.

– **Scoring Function Selection:** In this work, seven classification algorithms drawn from different families of machine learning models were tested for their ability to fit the training data and used as the team-fitness scoring function. While Random Forest yielded the best ROC for all the three datasets, there can be other algorithms that have not been validated and may perform better. Hence, *RECAST* is developed such that this part can be plugged in if better scoring functions are discovered in the future.

7 Related Work

Developing automatic systems that recommend software practitioners for a given software task has recently gained attention from research communities. However, most of such relevant systems aim to make a recommendation for particular single tasks, without taking into account that a software team is typically composed of members with diverse technical backgrounds and roles. Hence, in this section, we first discuss such relevant single-role recommendation algorithms for software development tasks. Since our goal is to develop a recommendation algorithm for whole software teams, we also discuss work on the relevant algorithms proposed in other domains of applications.

7.1 Single-Role Recommendation

Single-role recommendation mainly focuses on recommending one individual at a time for a given task, with an assumption that the task can be resolved by one person. As a result, a single-role recommendation lacks the ability to recommend a set of individuals with different skills and roles. In software engineering literature, previous studies have investigated the possibility to develop automatic single-role recommendation algorithms for developers, reviewers, integrators, etc.

Naguib et al. (2013) investigated the issues found in bug reports and proposed an algorithm to recommend assignees to fix given bugs. They proposed the solution to address this problem by creating the activity profiles for all users in the repository. An activity profile contains two parts, namely roles and topic associations. For a given bug report, the proposed system analyzes user activity profiles to indicate activities that each user has done in the past regarding bug reports. The proposed algorithm was evaluated against the LDA-SVM based assignee recommendation technique.

Thongtanunam et al. (2015) proposed RevFinder, a file location-based code-reviewer recommendation approach. Their approach is based on the intuition that codes that are stored in directories with similar file paths should be reviewed by the same person. Later, Yu et al. (2016) proposed a reviewer recommendation algorithm for pull-requests in GitHub. Their algorithms analyzed social relations between contributors and reviewers to generate the comment network. This comment network is combined with a traditional method to improve the reviewer's recommendation accuracy. Recently, Rahman et al. (2016) presented CoRReCT, a reviewer recommendation approach that considers not only the history of the relevant cross-projects but also the experience of a developer in certain specialized technologies associated with a pull request, to determine his/her ability to review code.

Liao et al. (2017) presented a topic-based code integrator recommendation algorithm for pull requests. Their approach is based on the ability to capture textual content from pull requests, which are used to train a topic model using LDA. The similarity between the two pull requests is then quantified using the similarity between the topic distribution between the two requests. Integrators who used to integrate similar pull-requests are then recommended. Recently, de Lima Júnior et al. (2018) also investigated different attributes that characterize integrators for pull requests, and used machine learning classification techniques to rank integrators to a given pull request. They evaluated the proposed algorithm on 32 open-source projects and illustrated that their method outperforms the state-of-the-art integrator recommendation method by 54%.

Besides software engineering, such a single-role recommendation methodology has been applied in many real-world domains. In human resource recruitment, Malinowski et al. (2008) presented a decision support system for selecting best fit team members using the HR information system. They focused on recommending a person who not only fits the job description but also fits the existing team. Different aspects of fitness were considered such as fitness to the job (P-J), fitness to the team (P-T), fitness to the environment (P-E), fitness to the organization (P-O), and person-vocation compatibility (P-V). As a result, they presented a recommendation approach which focuses on two dimensions. The first dimension concerns unary attributes such as skills and personality. The second involves relation attributes that determine how well a person fits the current team. We can adopt this approach to our project by incorporating social network analysis to measure the interactions and connectedness. Moreover, this research also provides a method to calculate the trust value between the candidate and all members.

Recently, Gupta et al. (2014) proposed a system that recommends potential members for a team based on user requests, including user specification of a specific job, and skills mapping to a job title. They created a user interface to provide a recommendation service to users. They mainly considered the candidates' profile data and activity data which reflects their behaviors. This recommendation system contains three modules: skill mapping, candidate identification, and notification. For skill mapping, the system matches candidate skills with the specification of the job titles. Each job title must specify the required skills and their important values, e.g. Java, HTML, PHP, etc. For candidate identification, they investigated candidate profiles, each of which is consisted of skills and experience. Keywords were used to distinguish expertise and skills from a job description or experience. Lastly, the notification module notifies the specific candidate when he/she is asked to join the project. We could adopt the use of the mapping technique that links between the ability of the candidate and the task description. However, to apply this technique to the software engineering field, we have to consider the fact that anyone can create an issue or a task.

While single-role recommendation techniques cannot be directly applied to solve team recommendation problems, they provide early insight into features that can be drawn from each candidate, that can be adapted to compute the aggregate features for the whole team. Especially in the software engineering domain, where collaborative software development has become a norm, integrating individual expertise to compose an effective team could prove crucial for large-scale software projects.

7.2 Team Recommendation

Different from single-role recommendation, team recommendation aims to generate configurations of team members taking into account diverse skills, roles, and collaboration history. Limited work has investigated techniques for team recommendation in collaborative software engineering. Hence, we discuss relevant techniques applied in other fields and applications.

Alberola et al. (2016) presented an artificial intelligence tool for heterogeneous team formation in the classroom. They formed teams by using Belbin's role theory and used artificial intelligence techniques to handle the uncertainty of the student roles. They also considered the information that was collected before and after working on a team. Applying this solution may not be suitable for the software development process since each developer would have to shoulder the burden of evaluating the team every time that a task is finished.

Hupa et al. (2010) introduced a scoring function for a team based on its members' social context and suitability to a particular project. They proposed a three-dimensional social network that was used to represent the social context of individuals. The network consists of a knowledge network, a trusted network, and an acquaintance network. The knowledge network is a bipartite graph with an edge connecting a person to his/her skill. The weight of an edge represents the experience level of the corresponding skill. The trust network is a directed graph with an edge connecting a person to 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 interaction of two people. The graphs are used to quantify the following features: Closeness Centrality, Average Interpersonal Trust Measure, Aggregated Skill Difference, Maximum Skill Distribution. After defining criteria that will be used to score a team on a project, they combine all criteria into one function and optimize it using the reference point method. The idea of a multidimensional graph can be applied to our problem setting; however, their method requires enumerating all possible team configurations, which is infeasible in large-scale collaboration communities.

Later, Datta et al. (2011) developed an expert recommendation system. This system allows users to input a set of required skills, then the system will form teams that comprise the necessary skills. Besides, the user can also adjust the recommended team members using several criteria, e.g. school, status, etc. Users can specify the importance of each skill and recommendation parameters, e.g. number of members in the team, cohesiveness importance, etc. To determine the quality of the team, the system considers two features: competence coverage which describes team skills; 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 applied to our problem to allow more flexible recommendations; however, this work does not mention how candidate teams are enumerated, which is important for large datasets.

Recently, Liu et al. (2014) proposed an approach to recommend teams that satisfy role requirements. They considered both individual and aggregate strengths as features. They developed a team strength function that was used to determine both compatibility and strength of an input team. They divided features into two types which are individual strengths and team features. Individual strength features are features that describe a person's strength without considering the team. The individual strength features comprise experience, win experience, win rate, and roles. On the other hand, the team features characterize the compatibility of a team, consisting of team closeness and social connection. The weights of features were optimized by a logistic regression model trained with historical project outcomes. Because it is infeasible to compute all the combinations of teams, they used the Max-Logit algorithm, a statistical learning-based algorithm studied in the potential game theory, to find the approximately best team. Their findings also support the expandability of their team formation algorithm. Our problem setting is similar to theirs; therefore, we can apply their idea of using the Max-Logit algorithm to search for the approximately best team. However, the scoring function and features would need to be revisited to make them specific to the software engineering domain.

8 Conclusions

Collaborative software development has emerged since the complexity of software systems is increasing. Multiple roles and diverse skills of teams are required to handle those software development tasks. In this work, we proposed a machine learning based recommendation model called, *RECAST*, that can suggest software teams suitable for software development tasks. Our approach outperforms the common baseline for a recommendation system and the existing team recommendation approach in our empirical evaluation on three open-source projects. The evaluation results also suggest that our approach also performs best in both recommending whole team members with suitable roles and recommending an individual team member for a specific role. In our future work, we aim to enhance our approach by applying advanced techniques such as deep learning models to improve recommendation performance. We also aim to evaluate our approach to commercial setting projects and conduct a user evaluation to study how our approach should be adopted by software development teams. Furthermore, the impacts of software team recommendation at the ecosystem level, in addition to the issue level, could be further investigated. Finally, geographical and community types of software teams, along with types of connections among them could be investigated for a possible extension of the current method.

Acknowledgements This research is supported by the Thailand Science Research and Innovation (TSRI), formerly known as Thailand Research Fund (TRF), through grant RSA6280105.

References

Al-Subaihin A, Sarro F, Black S, Capra L (2019) Empirical comparison of text-based mobile apps similarity measurement techniques. Empir Softw Eng 24(6):3290–3315

Alberola JM, Del Val E, Sanchez-Anguix V, Palomares A, Teruel MD (2016) An artificial intelligence tool for heterogeneous team formation in the classroom. Knowl-Based Syst 101:1–14

Alharbi AS, Li Y, Xu Y (2017) Integrating lda with clustering technique for relevance feature selection. In: Australasian joint conference on artificial intelligence. Springer, pp 274–286

Assavakamhaenghan N, Choetkiertikul M, Tuarob S, Kula RG, Hata H, Ragkhitwetsagul C, Sunetnanta T, Matsumoto K (2019) Software team member configurations: A study of team effectiveness in moodle. In: 2019 10th International workshop on empirical software engineering in practice (IWESEP). IEEE, pp 19–195

Asuncion A, Welling M, Smyth P, Teh YW (2009) On smoothing and inference for topic models. In: Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence, UAI '09. AUAI Press, Arlington, pp 27–34. ISBN 978-0-9749039-5-8. http://dl.acm.org/citation.cfm?id=1795114.1795118

Baj-Rogowska A (2017) Sentiment analysis of facebook posts: The uber case. In: Proceedings of the 8th International conference on intelligent computing and information systems (ICICIS). IEEE, pp 391–395

Blei DM, Ng AY, Jordan MI (2003) Latent dirichlet allocation. J Mach Learn Res 3(Jan):993–1022

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Cabanillas C, Resinas M, Mendling J, Ruiz-Cortés A (2015) Automated team selection and compliance checking in business processes. In: Proceedings of the international conference on software and system process. pp 42–51

Chen T-H, Thomas SW, Hassan AE (2016) A survey on the use of topic models when mining software repositories. Empir Softw Eng 21(5):1843–1919

Choetkiertikul M, Dam HK, Tran T, Ghose A (2017) Predicting the delay of issues with due dates in software projects. Empir Softw Eng 22(3):1223–1263. ISSN 15737616. https://doi.org/10.1007/s10664-016-9496-7

Coelho J, Valente MT (2017) Why modern open source projects fail. In: Proceedings of the 2017 11th Joint meeting on foundations of software engineering, pp 186–196

Dai H, Huang Y, Yang L (2015) Game theoretic max-logit learning approaches for joint base station selection and resource allocation in heterogeneous networks. IEEE J Select Areas Commun 33(6):1068–1081

Datta A, Tan Teck Yong J, Ventresque A (2011) T-recs: team recommendation system through expertise and cohesiveness. In: Proceedings of the 20th international conference companion on World wide web. pp 201–204

de Lima Júnior ML, Soares DM, Plastino A, Murta L (2018) Automatic assignment of integrators to pull requests: The importance of selecting appropriate attributes. J Syst Softw 144:181–196

Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R (1990) Indexing by latent semantic analysis. J Am Soc Inf Sci 41(6):391–407

Ferreira DJ, Caetano SS, Junior CGC (2017) An automatic group formation method to foster innovation in collaborative learning at workplace. Int J Innov Educ Res 5(4):28–43

Fox E (2008) Emotion science cognitive and neuroscientific approaches to understanding human emotions. Palgrave Macmillan, London

Gao D, Tong Y, She J, Song T, Chen L, Xu K (2017) Top-k team recommendation and its variants in spatial crowdsourcing. Data Sci Eng 2(2):136–150

Gharehyazie M, Filkov V (2017) Tracing distributed collaborative development in apache software foundation projects. Empir Softw Eng 22(4):1795–1830

Gharehyazie M, Posnett D, Vasilescu B, Filkov V (2015) Developer initiation and social interactions in oss: A case study of the apache software foundation. Empir Softw Eng 20(5):1318–1353

Grigore M, Rosenkranz C (2011) Increasing the willingness to collaborate online: an analysis of sentiment-driven interactions in peer content production. In: Galletta DF, Liang T (eds) Proceedings of the international conference on information systems, ICIS 2011, Shanghai, China, December 4-7, 2011. Association for Information Systems. http://aisel.aisnet.org/icis2011/proceedings/onlinecommunity/20

Gupta P, Xhabija F, Shoup MD, Brikman Y, Crosa A, Ramirez R (2014) Team member recommendation system. US Patent App. 13/907,577

Hassan S-U, Akram A, Haddawy P (2017) Identifying important citations using contextual information from full text. In: 2017 ACM/IEEE Joint conference on digital libraries (JCDL). IEEE, pp 1–8

Hofmann T (2001) Unsupervised learning by probabilistic latent semantic analysis. Mach Learn 42(1-2):177–196

Hogan JM, Thomas R (2005) Developing the software engineering team. pp 203–210

Hupa A, Rzadca K, Wierzbicki A, matchmaking A. Datta. (2010) Interdisciplinary Choosing collaborators by skill, acquaintance and trust. In: Computational social network analysis. Springer, pp 319–347

Iaffaldano G, Steinmacher I, Calefato F, Gerosa M, Lanubile F (2019) Why do developers take breaks from contributing to oss projects?: a preliminary analysis. In: Proceedings of the 2nd International workshop on software health. IEEE Press, pp 9–16

Jeni LA, Cohn JF, De La Torre F (2013) Facing imbalanced data–recommendations for the use of performance metrics. In: 2013 Humaine association conference on affective computing and intelligent interaction. IEEE, pp 245–251

Jiarpakdee J, Tantithamthavorn C, Hassan AE (2019) The impact of correlated metrics on the interpretation of defect models. IEEE Trans Softw Eng

Jiarpakdee J, Tantithamthavorn C, Treude C (2020) The impact of automated feature selection techniques on the interpretation of defect models. Empir Softw Eng :1–49

Kale A, Karandikar A, Kolari P, Java A, Joshi A (2007) Modeling trust and influence in the blogosphere using link polarity. In: Proceedings of the international conference on Weblogs and social media (ICWSM 2007)

Khoshgoftaar TM, Golawala M, Van Hulse J (2007) An empirical study of learning from imbalanced data using random forest. In: 19th IEEE International conference on tools with artificial intelligence (ICTAI 2007), vol 2. IEEE, pp 310–317

Li H, Chen T-HP, Shang W, Hassan AE (2018) Studying software logging using topic models. Empir Softw Eng 23(5):2655–2694

Liao Z, Li Y, He D, Wu J, Zhang Y, Fan X (2017) Topic-based integrator matching for pull request. In: Proceedings of GLOBECOM 2017-2017 IEEE global communications conference. IEEE, pp 1–6

Lindsjørn Y, Sjøberg DI, Dingsøyr T, Bergersen GR, Dybå T (2016) Teamwork quality and project success in software development: A survey of agile development teams. J Syst Softw 122:274–286

Liu H, Qiao M, Greenia D, Akkiraju R, Dill S, Nakamura T, Song Y, Nezhad HM (2014) A machine learning approach to combining individual strength and team features for team recommendation. In: Proceedings of the 13th International conference on machine learning and applications. IEEE, pp 213–218

Malinowski J, Weitzel T, Keim T (2008) Decision support for team staffing: An automated relational recommendation approach. Decis Support Syst 45(3):429–447

Manning CD, Schütze H, Raghavan P (2008) Introduction to information retrieval. Cambridge university press, Cambridge

McIntosh A, Hassan S, Hindle A (2019) What can android mobile app developers do about the energy consumption of machine learning? Empir Softw Eng 24(2):562–601

Misra H, Cappé O, Yvon F (2008) Using lda to detect semantically incoherent documents. In: CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning, pages 41–48

Mistrík I, Grundy J, Van der Hoek A, Whitehead J (2010) Collaborative software engineering: challenges and prospects. Berlin, Springer, pp 389–403

Moe NB, Dingsøyr T, Dybå T (2010) A teamwork model for understanding an agile team: A case study of a Scrum project. Inf Softw Technol 52(5):480–491. ISSN 09505849. https://doi.org/10.1016/j.infsof.2009.11.004

Monderer D, Shapley LS (1996) Potential games. Games Econ Behav 14(1):124–143

Naguib H, Narayan N, Brügge B, Helal D (2013) Bug report assignee recommendation using activity profiles. In: Proceedings of the 10th working conference on mining software repositories (MSR). IEEE, pp 22–30

Newman D, Lau J, Grieser K, Baldwin T (2010) Automatic evaluation of topic coherence. inhuman language technologies: The 2010 annual conference of the north american chapter of the association for computational linguistics hlt'10

Osborne MJ, Rubinstein A (1994) A course in game theory, MIT press, Cambridge

Ouni A, Kula RG, Inoue K (2016) Search-based peer reviewers recommendation in modern code review. In: Proceedings of IEEE International conference on software maintenance and evolution (ICSME). IEEE, pp 367–377

Petkovic D, Sosnick-Pérez M, Huang S, Todtenhoefer R, Okada K, Arora S, Sreenivasen R, Flores L, Dubey S (2014) Setap: Software engineering teamwork assessment and prediction using machine learning. In: Proceedings of IEEE Frontiers in education conference (FIE) proceedings. IEEE, pp 1–8

Ragkhitwetsagul C, Krinke J (2019) Siamese: scalable and incremental code clone search via multiple code representations. Empir Softw Eng 24(4):2236–2284

Rahman MM, Roy CK, Collins JA (2016) Correct: code reviewer recommendation in github based on cross-project and technology experience. In: Proceedings of the 38th International conference on software engineering companion. pp 222–231

Ramage D, Hall D, Nallapati R, Manning CD (2009) Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In: Proceedings of conference on empirical methods in natural language processing: Volume 1-Volume 1. Association for Computational Linguistics, pp 248–256

Rosen C, Shihab E (2016) What are mobile developers asking about? a large scale study using stack overflow. Empir Softw Eng 21(3):1192–1223

Song Y, Wong SH, Lee. K-W (2011) Optimal gateway selection in multi-domain wireless networks: A potential game perspective. In: Proceedings of the 17th annual international conference on Mobile computing and networking. pp 325–336

Steyvers M, Griffiths T (2007) Probabilistic topic models. In: Handbook of latent semantic analysis, vol 427, pp 424–440

Sudhakar GP (2012) A model of critical success factors for software projects. J Enterp Inf Manag

Surian D, Liu N, Lo D, Tong H, Lim E-P, Faloutsos C (2011) Recommending people in developers' collaboration network. In: Proceedings of the 18th Working conference on reverse engineering. IEEE, pp 379–388

Tantithamthavorn C, Hassan AE, Matsumoto K (2018) The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Trans Softw Eng

Thelwall M, Buckley K, Paltoglou G, Cai D, Kappas A (2010) Sentiment strength detection in short informal text. J Am Soc Inf Sci Technol 61(12):2544–2558

Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto K-I (2015) Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In: Proceedings of IEEE 22nd International conference on software analysis, evolution, and reengineering (SANER). IEEE, pp 141–150

Tuarob S, Pouchard LC, Mitra P, Giles CL (2015) A generalized topic modeling approach for automatic document annotation. Int J Digit Libr 16(2):111–128

Tuarob S, Strong R, Chandra A, Tucker CS (2018) Discovering discontinuity in big financial transaction data. ACM Trans Manag Inf Syst (TMIS) 9(1):1–26

Tuarob S, Kang SW, Wettayakorn P, Pornprasit C, Sachati T, Hassan SU, Haddawy P (2020) Automatic classification of algorithm citation functions in scientific literature. IEEE Trans Knowl Data Eng 32(10):1881–1896

Xia X, Lo D, Wang X, Zhou B (2013) Accurate developer recommendation for bug resolution. In: 2013 20th Working conference on reverse engineering (WCRE). IEEE, pp 72–81

Xia X, Lo D, Wang X, Zhou B (2015) Dual analysis for recommending developers to resolve bugs. J Softw Evol Process 27(3):195–220

Yang H, Sun X, Li B, Hu J (2016) Recommending developers with supplementary information for issue request resolution. In: Proceedings of the 38th International conference on software engineering companion. pp 707–709

Yi H, Xiong Q, Zou Q, Xu R, Wang K, Gao M (2019) A novel random forest and its application on classification of air quality. In: 2019 8th International congress on advanced applied informatics (IIAI-AAI). IEEE, pp 35–38

Yu Y, Wang H, Yin G, Wang T (2016) Reviewer recommendation for pull-requests in github What can we learn from code review and bug assignment? Inf Softw Technol 74:204–218

Zanjani MB, Kagdi H, Bird C (2015) Automatically recommending peer reviewers in modern code review. IEEE Trans Softw Eng 42(6):530–543

Zhang Z, Ren W, Yang Z, Wen G (2020a) Real-time seam defect identification for al alloys in robotic arc welding using optical spectroscopy and integrating learning. Measurement 156:107546

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

Zhang Z-W, Jing X-Y, Wang T-J (2017) Label propagation based semi-supervised learning for software defect prediction. Autom Softw Eng 24(1):47–69

Suppawong Tuarob received the BSE and MSE degrees both in computer science and engineering from the University of Michigan-Ann Arbor and the MS degree in industrial engineering and the PhD degree in computer science and engineering both from the Pennsylvania State University. Currently, he is an assistant professor of computer science at Mahidol University, Thailand. His research involves data mining and knowledge discovery in large-scale software engineering, scholarly, social media, and healthcare domains by applying multiple cutting-edge techniques, such as machine learning, topic modeling, and sentiment analysis.

Noppadol Assavakamhaenghan received BSc in Information and Communication Technology from Mahidol University, Thailand. He is currently studying for an MSc in Information Science and Technology at Nara Institute of Science and Technology, Japan. His research interests include machine learning and knowledge discovery in software processes and software ecosystems.

Waralee Tanaphantaruk earned her bachelor's degree in Information and Communication Technology from Mahidol University, Thailand. Her major is the database and intelligence System track. Her area of interests includes Recommendation Systems and Information Retrieval.

Ponlakit Suwanworaboon received BSc in Information and Communication Technology from Mahidol University. He is proficient in the areas of web application penetration test and vulnerability assessment. His current research interests include Cyber Security, Cloud Computing and Artificial Intelligence.

Saeed-Ul Hassan is the Director of Artificial Intelligence Lab and a faculty member at Information Technology University (ITU) in Pakistan, a former Post-Doctorate Fellow at the United Nations University – with more than 15 years of hands-on experience of advanced statistical techniques, artificial intelligence, and software development client work. He earned his Ph.D. in the field Information Management from Asian Institute of Technology. He has also served as a Research Fellow at National Institute of Informatics in Japan. Dr. Saeed's research interests lie within the areas of Data Science, Artificial Intelligence, Scientometrics, Information Retrieval and Text Mining. Dr. Hassan is also the recipient of James A. Linen III Memorial Award in recognition of his outstanding academic performance. More recently, he has been awarded Eugene Garfield Honorable Mention Award for Innovation in Citation Analysis by Clarivate Analytics, Thomson Reuters.

Morakot Choetkiertikul is a lecturer at the Faculty of Information and Communication Technology (ICT), Mahidol University, Thailand where he co-founded the Software Engineering Research Unit (SERU). He graduated Ph.D. in computer science from the University of Wollongong (UOW), Australia. His research interests include applying AI solutions to improve software quality and software process. His research has been published at top-tier software engineering venues, such as IEEE Transactions on Software Engineering (TSE), Empirical Software Engineering (EMSE), and the International Conference on Software Engineering (ICSE). More info: https://muict-seru.github.io/

Affiliations

Suppawong Tuarob[1] · Noppadol Assavakamhaenghan[1] · Waralee Tanaphantaruk[1] · Ponlakit Suwanworaboon[1] · Saeed-Ul Hassan[2] · Morakot Choetkiertikul[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

Saeed-Ul Hassan
saeed-ul-hassan@itu.edu.pk

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

2 Information Technology University, Lahore, Pakistan