# Skill-based Team Formation in Software Ecosystems

**Daniel Schall** [1]

**Abstract.** This paper introduces novel techniques for the discovery and formation of teams in software ecosystems. Formation techniques have a wide range of applications including the assembly of expert teams in open development ecosystems, finding optimal teams for ad-hoc tasks in large enterprises, or working on complex tasks in crowdsourcing environments. Software development performance and software quality are affected by the skills and application domain experiences that the team members bring to the project. Team formation in software ecosystems poses new challenges because development activities are no longer coordinated by a single organization but rather evolve much more flexibly within communities. A suitable approach for finding optimal teams must consider expertise, user load, social distance and collaboration cost of team members. We have designed this model specifically for the analysis of large-scale software ecosystems wherein users perform development activities. We have studied our approach by analysing the R ecosystem and find that our approach is well suited for the team discovery in software ecosystems.

## 1 Introduction

Establishing a software ecosystem becomes increasingly important for a companies' collaboration strategy with other companies, open source developers and end users. The idea behind software ecosystems differs from traditional outsourcing techniques [19, 22]. The initiating actor does not necessarily own the software produced by the contributing actors nor are contributing actors hired by the initiator (e.g., a firm). All actors as well as software artefacts, however, coexist in an interdependent way. For example, actors jointly develop applications and thus there is a relationship among the actors. Software components may depend on each and thus there is a relationship among the components. This is a parallel to natural ecosystems where the different members of the ecosystems (e.g., the plants, animals, or insects) are part of a food network where the existence of one species depends on the rest. In contrast to natural ecosystems, some software ecosystems may be mainly top-down controlled, with most changes driven by change requests and bug reports coming from other actors [20]. Other software ecosystems may be controlled in a bottom-up manner, primarily driven by input from its core developers [21].

[1] Siemens Corporate Technology, Siemensstrasse 90, 1211 Vienna, Austria, email: daniel.schall@siemens.com

A key challenge in software ecosystems is to manage quality of software [8, 9, 15] and addressing nonfunctional requirements (NFRs) in general [4]. Software vendors may require their plug-in developers to maintain certain quality levels to deserve an approved status. As shown in earlier research, individual as well as development team skill has a significant effect on the quality of a software product [5, 7, 12]. The approach in this work takes a socio-technical view on software ecosystems wherein ecosystems are understood as the interplay between the social system and the technical system [11, 13]. Software development teams composed of members with prior joint project experience may be more effective in coordinating programmers' distributed expertise because they have developed knowledge of who knows what [12]. This paper addresses the problem of team formation in open, dynamic software ecosystems. In open source development teams are more often than not formed spontaneously (i.e., they 'emerge') based on people's availability, willingness to collaborate and to contribute to a certain task.

Potential tasks for expert teams in software ecosystems and open source development include:

- Come up with a software design and/or implementation of a complex component or subsystem.
- Perform software architecture review of an existing implementation or provide expert opinion about an emerging technology.

To give a concrete example of a potential high-level task, a complex design or implementation may involve the analysis of timeseries data including data extraction from a source system, transformation of the data, storage, processing, and visualization. Clearly, this task typically requires multiple people with distinct skills such as data modelling, statistical knowledge (uni-/multivariate data analysis), data persistence management, and data visualization using various technologies and toolkits. Indeed, the high-level task needs to be further decomposed into smaller task. The goals of this work is to find a team of experts given a set of high level skills. Once the team has been discovered, detailed task decomposition and work planning can be performed, which is however not in the focus of this work.

We provide the following key contributions:

- A novel approach supporting team formation in software ecosystems based on user expertise, load, social distance among team members, and collaboration cost.
- Support the discovery of potential mediators if social connectedness in teams is low.

- Analysis of user expertise to recommend the most suitable team members.
- Evaluation of the concepts using data collected from the Comprehensive R Archive Network (CRAN) - the largest public repository of R packages.

The remainder of this paper is organized as follows. In Section 2 we overview related work and concepts. In Section 3 we introduce our team formation approach. Experiments are detailed in Section 4. The paper is concluded in Section 5.

## 2 Related Work

The success of a project depends not only on the expertise of the people who are involved, but also on how effectively they collaborate, communicate and work together as a team [17, 26]. On the one hand, a team must contain the right set of expertise, but on the other hand one should determine a staffing level that, while comprising all the needed expertise, minimizes the cost and contributes to meeting the project deadline [10]. The most critical resource for knowledge teams is expertise and specialized skills in using and handling tools. But the mere presence of expertise in a team is insufficient to produce high-quality work. A team must collaborate in an effective manner. It has been found that prior collaborative ties have a profound effect on developers' project joining decisions [12].

In software engineering, team formation is often needed to perform a development or maintenance activity. A general trend is the growing number of large scale software projects, software development and maintenance activities demanding for the participation of larger groups [6, 14]. In [3], the authors proposed assignment of experts to handle bug reports based on previous activity of the expert. Social collaboration on GitHub including team formation has been addressed in [18]. The authors [2] study the problem of online team formation. In their work, a setting in which people possess different skills and compatibility among potential team members is modelled by a social network. It has to be noted that team formation in social networks is an *NP*-hard problem. Thus, optimization algorithms such as genetic algorithms [29] should be considered in solving the team composition problem [31].

Expertise identification is one of the key challanges and success factors for team work and collaboration [16]. The discovery of experts is becoming critical to ease the communication between developers in case of global software development or to better know members of large software communities [30]. Network analysis techniques offer a rich set of theories and tools to analyse the topological features and human behaviour in online communities. We have extensively studied the automatic extraction of expertise profiles in our prior research (see [24, 25, 27]) and build upon our social network based expertise mining framework.

With regards to team formation in open source communities as well as software ecosystems, there is still a gap in related work and to our best knowledge there is no existing approach that supports formation based on mined expertise profiles.

## 3 Formation in Ecosystems

Here we present the overall team formation algorithm. We employ a genetic algorithm that attempts to find the best team. Genetic algorithms (GAs) mimic Darwinian forces of natural selection to find optimal values of some function [23].

For each team a single number denoting the team's fitness is calculated (where larger values are better).

### 3.1 Genetic Algorithm Outline

There are multiple objectives that need to be optimized. The objectives are to:

- maximize the average expertise score for given skills
- minimize the average cost
- minimize the average distance

The team with the best trade-off among these objectives shall obtain the highest fitness and will be recommended as the best fitting team. The required team skills are stated by customers who wish to assign a specific complex task to a team of experts — be it within a corporation or outsourcing a specific task to the crowd. Our assumption is that such complex tasks demand for the expertise of multiple team members. Within our formation approach, additional constraints can be considered such as one person shall only cover one skill and not multiple ones. Indeed, in practice people are familiar with multiple topics thereby covering multiple skills. Factors such as matching user load with complexity, effort, and deadline of a task are not in focus of this work. The reader may refer to [25] for further information on these topics.

The main computational steps of our formation approach are introduced and elaborated in detail in Algorithm 1. The relevant lines in Algorithm 1 are specified in parenthesis.

1. Based on the set $S = \{s_1, s_2, \ldots, s_n\}$ of demanded skills, prepare a mapping structure that holds skills, users $U$, and expertise ranking scores (lines 6-9). In this step, current user load is evaluated (based on previously assigned tasks) and users with high load are filtered out.
2. Initialize a population of individuals. An individual is a team consisting of $n$ team members where $n$ can be configured. The parameter $n$ is given by the size of the skill set $S$ if each team member has to provision exactly one skill (line 12).
3. Loop until max iterations have been reached and compute the main portion of the genetic algorithm based search heuristic. Finally, after this loop select the team with the highest fitness (lines 14-48).
4. Depending on the ecosystems community structure and the demanded set of skills, teams may have good or poor connectivity in terms of social links among team members. Therefore, construct a subgraph of the social collaboration graph $G_S$ containing only the nodes from the best team and their edges between each other. Analyse the connectivity of this subgraph by computing the average number of neighbours (lines 50-51).
5. If connectivity is low, try to find a dedicated coordinator who is ideally connected to all team members through social links. The role of the coordinator is to mediate communication among members and strengthen team cohesion (line 53-56).

**Algorithm 1** Formation algorithm.

```
1: input: skills S, population size p_size,
2: elitism elitism_k, max iterations max_iter
3: output: best individual - team with the highest fitness
4: # init mappings of skills, users, scores
5: M ← ∅
6: for Skill s ∈ S do
7:     m ← getRankingScoreMapping(s)
8:     M[s] ← m # add mapping
9: end for
10: # initialize population of
11: # randomly composed individuals
12: P ← createPopulation(S, M, p_size)
13: iter ← 0 # iteration counter
14: while iter++ < max_iter do
15:     # new population
16:     P' ← createEmptyPopulation(p_size)
17:     # obtain a ranked list
18:     R ← rankPopulationByFitness(P)
19:     # elitism - copy small part of the fittest
20:     for i = 0 ... elitism_k − 1 do
21:         # add to population
22:         P'[i] ← getIndividualByRankIndex(R, i)
23:     end for
24:     # build new population
25:     i ← elitism_k
26:     while i < p_size do
27:         # stochastically select from P
28:         I[] ← rouletteWheelSelection(P)
29:         # crossover
30:         if randomNumber < crossover_rate then
31:             # assign new offspring
32:             I[0] ← crossover(I[0], I[1])
33:         end if
34:         # mutation
35:         if randomNumber < mutation_rate then
36:             # assign mutated individual
37:             I[0] ← mutate(I[0])
38:         end if
39:         # add new offspring
40:         P'[i++] ← I[0]
41:     end while
42:     # assign the new population
43:     P ← P'
44:     # evaluate population - total fitness
45:     evaluate(P)
46: end while
47: # this is the best team
48: I ← findBestIndividual(P)
49: # construct a graph G_I based on I
50: G_I ← extractSubGraph(G_S, I)
51: if checkConnectivity(G_I) < ζ then
52:     # find node to increase connectivity
53:     u ← findCoordinator(G_I)
54:     if u ≠ null then
55:         addNode(G_I, u)
56:     end if
57: end if
58: # extract the node set N_I ⊂ U from G_I
59: N_I ← getNodes(G_I)
60: return N_I # node set of best team members
```

## 3.2 Detailed Computational Steps

In the following we detail the steps in the algorithm and explain additional functions that are invoked while executing the formation algorithm.

### *3.2.1 Rank Expertise by Skills*

Expertise profiles are not created in a predefined, static manner. Expertise is calculated based on actual community contributions. However, we do not attempt to analyse detailed user contributions in terms of software versioning control systems but rather focus on 'high-level' package metadata thereby following a less privacy intrusive approach. The method used for determining the expertise scores is not within the focus of this work due to space limits. The interested reader may refer to [27, 28] for information on the basic approach.

### *3.2.2 Basic Selection Strategies*

An initial set of candidate solutions are created and their corresponding fitness values are calculated. This set of solutions is referred to as a population and each solution as an individual (i.e., the team composed of team members). The individuals with the best fitness values are selected and combined randomly to produce offsprings, which make up the next population. The approach of selecting a subset of individuals with the best fitness values is called *elitism*. Elitism is realized by copying the fittest individuals to the next population.

To maintain a demanded population of individuals, individuals are selected and undergo *crossover* (mimicking genetic reproduction). For the team formation approach this means that team members are swapped between two teams. The basic part of the selection process is to stochastically select from one generation to create the basis of the next generation (see *rouletteWheelSelection* in line 28). The requirement is that the fittest individuals have a greater chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward. Weaker individuals are not without a chance. In nature such individuals may have genetic coding that may prove useful to future generations [1].

Individuals are also subject to random *mutations*. However, the probability of mutation is low because otherwise the genetic algorithm would be just a random search procedure. In our work we apply a 'smart' approach to mutation and do not select a replacement team member at random. Rather, a new team member is predicted and voting is performed by the existing team members.

### *3.2.3 Relationship-driven Mutation*

In traditional GAs, which are agnostic to the underlying nature of the population, mutation takes place by exchanging a gene by a random gene thereby maintaining genetic diversity. Here we perform prediction of edges between existing team members and newly randomly selected team members. If a threshold is surpassed, the randomly suggested team members is added to the team. This prediction approach ensures a much higher likelihood that the new team member will work within the team more effectively. We propose random forests for predicting edges between pairs of users. Random forests are a simple yet effective and robust method for classification problems. Prediction of edges between pairs of users is performed by modelling features describing the relationship between two nodes $u$ and $v$. At a high level, these features include common neighbours, the jaccard similarity index, joint community interest, and joint package dependencies.

### 3.2.4 Fitness Function

The last important ingredient of our GA based approach is the design of a workable fitness function. A fitness function is a type of objective function that is used to provide a single number. The idea is to discard 'bad' team configurations (i.e., individuals) and to breed new ones from the good configurations. The search heuristic is terminated when either the overall fitness converges or the maximum number of iterations have been reached.

The fitness function computes the value $\Phi_I$ as

$$\Phi_I = w_1 * expertise + w_2 * cost + w_3 * distance \quad (1)$$

where $w_1 + w_2 + w_3 = 1$. Each of the input factors $expertise, cost, distance$ needs to be scaled between 0 and 1. $\Phi_I$ is computed when invoking the function *evaluate* (see Algorithm 1 line 45). Later on $\Phi_I$ is used to rank the population by fitness (see Algorithm 1 line 17) as well as when calling $findBestIndividual$ (see Algorithm 1 line 48).

Algorithm 2 details the computational steps of an individual $I$'s fitness as defined by Eq. 1. An approximation of a cost factor is provided by community popularity in terms of number of neighbours in the social graph $G_S$. Thus, according to this logic more popular users are also more expensive. A perfect fitness score, given a set of skills $S$, would be 1. This is however impossible to achieve because expertise is also influenced by the user degree in $G_S$. Thus, a suitable tradeoff among these factors has to be found. The individual with the highest fitness within the population is then selected and recommended as the best team.

**Algorithm 2** Fitness function.

1: **input:** skills $S$, individual $I$, ranking score mapping $M$
2: **output:** fitness value $\Phi_I \in [0, 1]$ of individual $I$
3: $metrics \leftarrow \emptyset$ `# metrics as basis for fitness`
4: $score \leftarrow 0$ `# team expertise score`
5: $popularity \leftarrow 0$ `# popularity - to approximate cost`
6: **for** Skill $s \in S$ **do**
7: $u \leftarrow I[s]$ `# get member by skill`
8: $r_s \leftarrow M[s][u]$ `# expertise score by skill`
9: `# perform feature scaling`
10: $r'_s \leftarrow 1 - \frac{max(M[s]) - r_s}{max(M[s]) - min(M[s])}$
11: $score \leftarrow score + r'_s$
12: `# get user degree in` $G_S$
13: $k_u \leftarrow degree(G_S, u)$
14: `# perform feature scaling`
15: $k'_u \leftarrow \frac{k_{max} - k_u}{k_{max}}$
16: $popularity \leftarrow popularity + k'_u$
17: **end for**
18: `# add average team expertise score to metrics`
19: $addMetric(metrics, score/|S|)$
20: `# add average popularity to metrics`
21: `# higher community popularity means higher cost`
22: $addMetric(metrics, popularity/|S|)$
23: $dist \leftarrow 0$ `# social distance`
24: $Q \leftarrow queue(I)$
25: **while** $Q \neq \emptyset$ **do**
26: $u \leftarrow poll(Q)$
27: **for** $v \in Q$ **do**
28: `# unweighted shortest path distance`
29: `#` $d(u, v)$ `computed in` $G_S$
30: $d_{uv} = d(u, v)$
31: **if** $d_{uv} \neq null$ **then**
32: `# perform feature scaling`
33: `# lower values are better`
34: `#` $max(G_S)$ `is diameter of graph`
35: $d'_{uv} \leftarrow \frac{max(G_S) - d_{uv}}{max(G_S) - 1}$
36: $dist \leftarrow dist + d'_{uv}$
37: **end if**
38: **end for**
39: **end while**
40: $G_I \leftarrow extractSubGraph(G_S, I)$
41: `# add average distance to metrics`
42: $addMetric(metrics, dist/|edges(G_I)|)$
43: $\Phi_I \leftarrow 0$
44: **for** $m \in metrics$ **do**
45: $\Phi_I \leftarrow \Phi_I + w_m * metric$
46: **end for**

### 3.2.5 Coordinators

Based on the set of demanded skills and community structure, it may not be possible to find teams with good connectivity among the team members. The last steps in Algorithm 1 would be to check the connectivity of $I$ and to find a dedicated node who is ideally connected to all nodes in $I$ to mediate communication. All nodes in $U$ matching this constraint are then ranked based on their averaged expertise given the skill set $S$. The discovered node acting is potential team coordinator is added to the final team.

## 4 Experimental Evaluation

### 4.1 R Ecosystem

In this section we present our experiments. We focus on one part of the R ecosystem called the Comprehensive R Archive Network (CRAN). Other R-based communities not considered in this research are, for example, Bioconductor[2]. We have implemented a Web crawler to download[3] and parse R software package meta information available as HTML pages. This information includes contributing authors and package dependencies. In addition, CRAN provides so called CRAN task views, which are used in our analysis as skill or topic information. As an example, a given task view *Bayesian Inference* would be a single skill.

Figure 1 shows the package authorship graph $G_A$.

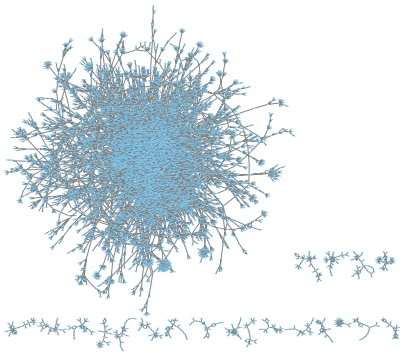

**Figure 1.** Authorship graph (subset).

[2] `https://www.bioconductor.org`
[3] `https://cran.r-project.org`, accessed on June 2016

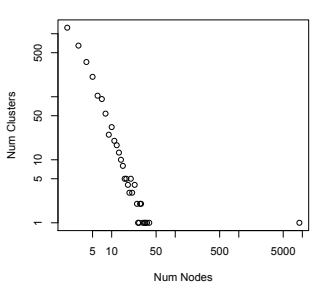


**Figure 2.** Clusters.

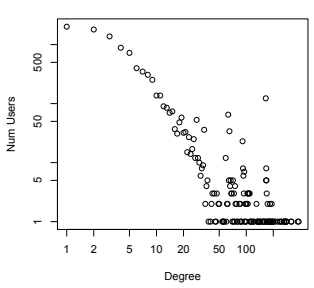


**Figure 3.** Degree $G_S$.

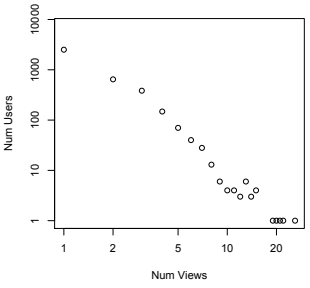


**Figure 4.** Views.

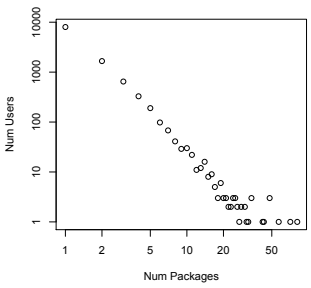


**Figure 5.** Packages.

In Fig. 1, only a subset is shown due to space limits. There are many more small clusters as those at the bottom of the figure. The community has one large cluster, the largest connected component (LCC) of the graph, containing 8985 nodes which are either users or packages. There are many smaller clusters with few users contributing to packages and also many users that contribute only to one single package.

Figure 2 shows the number of clusters versus the number of nodes in it (log-log scale). The LCC is depicted by the dot at the very bottom right corner of the figure. The majority of clusters has only few or just a single user package tuple. This also means that only users within the single largest connected component will be relevant for our analysis. Since we heavily rely on user degree in the social graph $G_S$, many users in the small clusters will have very low importance.

Figure 3 shows the degree distribution of the user graph $G_S$. Low degree of many users is explained by the large number of small clusters, which are mainly individual contributors of single packages. The graph $G_S$ consists of 11189 users. A fraction of 14% has a degree of 0 and 60% of users have a degree smaller or equal 3. The median[4] degree is 85. A fraction of 0.7% of users (74 users) have a degree larger than 85. Such degree distributions are typical in online communities.

The next Fig. 4 shows the relationship between CRAN task views and users. These views are interpreted as skills and expertise ranking is performed within the context of individual task views. In total, 33 views exist. 7316 users are not associated with any view (because their packages are not listed in any view). Thus, those users will not be considered in the formation algorithm. 3873 users are associated with one or more views. The median value for the number of views within this user segment is 11. This provides already a good diversity in terms of users having different skills.

The next step is to analyse the relationship between users and software packages. Figure 5 shows the number of users over packages. The median value for the number of software packages is 20. The dependencies of packages are visualized by Fig. 6. 4550 packages have exactly one dependency (the R environment). The median value is 10.

Finally, Fig. 7 shows the average number of dependencies by the number of users. The median value for the average number of dependencies is clearly 2.

[4] The median is used to separate the higher half of the user population from the lower half.

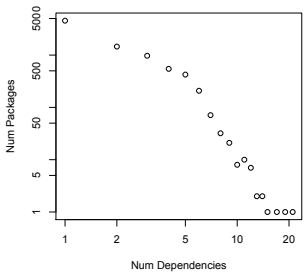


**Figure 6.** Dependencies.

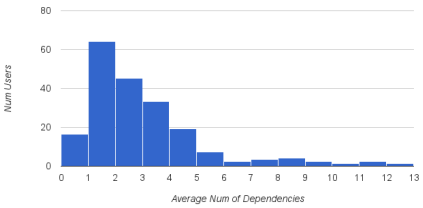


**Figure 7.** User dependencies.

## 4.2 Experiments Setup

In our experiments, we sampled a random set of CRAN task views representing the demanded team skills. The crossover probability is set to 0.7 and the mutation probability to 0.05. The metric weights for fitness calculation are set to $w_{score} = w_{cost} = w_{distance} = \frac{1}{3}$. In each experiment run, a population of 200 individuals plus 5 individuals for elitism has been created. We evaluate the quality of the expertise mining approach by checking key metrics such as degree of a user, number of packages in a given view (where the user is top-ranked), and number of all packages.

## 4.3 Qualitative Evidence

A team with the highest fitness for 5 skills is depicted by Fig. 8 and detailed in Table 1. The team has an average expertise score of 0.8, a cost of 0.6, and distance 1.0. These are excellent values. A perfect fitness of 1.0 is not attainable because there is a tradeoff between expertise and cost. Indeed, the maximum achievable fitness depends on the selected skills.

**Table 1.** Example of team recommendation for 5 skills.

| Skill | User | Rank | Score | Score ($G_S$) | Ch. | Deg. ($G_S$) | Pkg | All Pkg | Deg. (T) |
|---|---|---|---|---|---|---|---|---|---|
| Num. Math. | A. Gebhardt | 9 | 0.07681 | 0.00027 | 201 | 36 | 2 | 11 | 4 |
| Bayesian | M. Mächler | 1 | 0.87039 | 0.00150 | 0 | 276 | 1 | 57 | 4 |
| Meta Analysis | T. Lumley | 2 | 0.36787 | 0.00060 | 21 | 86 | 4 | 34 | 4 |
| Social Sciences | W. N. Venables | 8 | 0.19568 | 0.00021 | 509 | 50 | 5 | 10 | 4 |
| Psychometrics | K. Hornik | 1 | 0.85807 | 0.00113 | 4 | 318 | 3 | 71 | 4 |

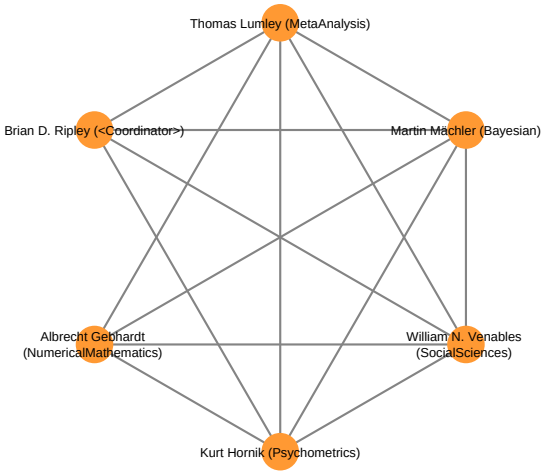


**Figure 8.** Example team spanning 5 skills.

Table 1 shows further team member details. **Rank** is the user rank within the specific task view (community rank for the given skill). **Score** is the numeric ranking score based on our advanced expertise mining model and **Score** ($G_S$) is the ranking score computed in $G_S$ using a standard PageRank. This comparison essentially demonstrates the impact of our advanced context-based ranking model (see [27, 28]). **Ch.** is the ranking change (context-based vs. standard PageRank). One can see that context has a high impact in terms of ranking position. We positively validated these results by checking the online profiles of the top-ranked users because most users have public Web sites. **Deg.** ($G_S$) depicts the degree (number of co-authors) in $G_S$. High degree typically means high community standing. **Pkg** depicts the number of packages in a given view and **All Pkg** all packages of the given user. **Deg. (T)** is the team degree. Here we see a perfectly connected team where each member is connected to all other members. The user 'Brian D. Ripley' is selected as the coordinator.

## 4.4 Performance Evaluation

We show the convergence of fitness values for both entire populations (depicted as $P$-fitness in Fig. 9) and for the best individuals in each population (depicted as $I$-fitness in Fig. 10). Convergence means that no major changes between one iteration to the next iteration are observed. 5 skills have been selected randomly. Different color codes depict 5 runs of GA formation algorithm. The best teams were identified after 7 iterations. Population fitness started to settle at 10.

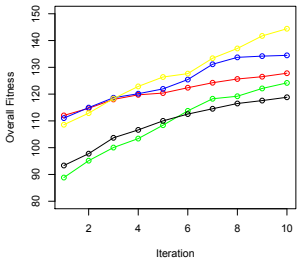


**Figure 9.** $P$-fitness.

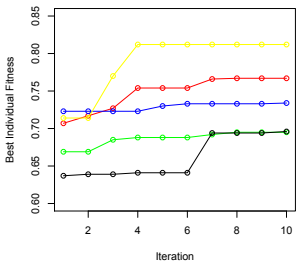


**Figure 10.** $I$-fitness.

In the following we answer the question whether an increasing number of skills results in more disconnected teams. Fig. 11 compares different populations with an increasing number of skills (from 5 to 9 skills depicted by S5 to S9).

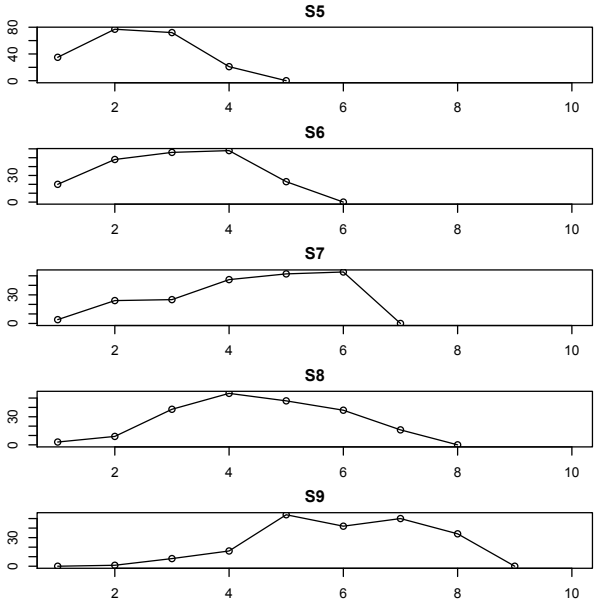


**Figure 11.** Skills vs. number of disconnected team.

The x-axis shows the number of disconnected team members and the y-axis the number of teams within the population (total number of populations is 205). The figures show that by increasing the number of skills the number of teams where only few members are disconnected decreases. More skills to be satisfied by individual users increases the chance that team members are disconnected from the rest of the team.

## 5 Conclusions

This work introduced team formation mechanisms for software ecosystems. We apply a genetic algorithm including a novel extension called relationship-driven mutation. In development teams, performance and quality are affected by the programming skills and domain experiences of the project's team members. We apply an advanced expertise mining approach to address this problem. Empirical results confirm the applicability of our presented methods.

Future work includes the following aspects. Estimating collaboration cost is not trivial and may depend on many factors, such as physical co-location, geographical distribution (including issues related to time difference), or cultural factors. We will analyse and model cost of collaboration. We will perform further evaluations of the approach in two directions. First, we want to validate results by engaging community members of the R ecosystem to get direct feedback on formation and expertise ranking results. Second, we will broaden team formation experiments for other types of software ecosystems including industrial and company internal software ecosystems. An additional area of investigation is the consideration of formal skill frameworks such as SFIA[5].

## REFERENCES


[1] Newcastle University, Roulette wheel selection. http://goo.gl/5CGi8t, Jan. 2007.

[2] Aris Anagnostopoulos, Luca Becchetti, Carlos Castillo, Aristides Gionis, and Stefano Leonardi, 'Online team formation in social networks', in *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pp. 839–848, New York, NY, USA, (2012). ACM.

[3] John Anvik and Gail C. Murphy, 'Reducing the effort of bug report triage: Recommenders for development-oriented decisions', *ACM Trans. Softw. Eng. Methodol.*, **20**(3), 10:1–10:35, (August 2011).

[4] Jakob Axelsson and Mats Skoglund, 'Quality assurance in software ecosystems: A systematic literature mapping and research agenda', *Journal of Systems and Software*, **114**, 69 – 81, (2016).

[5] Justin M. Beaver and Guy A. Schiavone, 'The effects of development team skill on software product quality', *SIGSOFT Softw. Eng. Notes*, **31**(3), 1–5, (May 2006).

[6] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu, 'Latent social structure in open source projects', in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '08/FSE-16, pp. 24–35, New York, NY, USA, (2008). ACM.

[7] Barry W. Boehm, 'Improving software productivity', *Computer*, 43–47, (1987).

[8] Jan Bosch, 'From software product lines to software ecosystems', in *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pp. 111–119, Pittsburgh, PA, USA, (2009). Carnegie Mellon University.

[9] M. Claes, T. Mens, and P. Grosjean, 'On the maintainability of cran packages', in *Software Maintenance, Reengineering and Reverse Engineering, 2014 Software Evolution Week*, pp. 308–312, (Feb 2014).

[10] Massimiliano Di Penta, Mark Harman, and Giuliano Antoniol, 'The use of search-based optimization techniques to schedule and staff software projects: An approach and an empirical study', *Softw. Pract. Exper.*, **41**(5), 495–519, (April 2011).

[11] R.P. dos Santos and C.M.L. Werner, 'Treating social dimension in software ecosystems through reuseecos approach', in *Digital Ecosystems Technologies (DEST), 2012 6th IEEE International Conference on*, pp. 1–6, (June 2012).

[12] Jungpil Hahn, Jae Y. Moon, and Chen Zhang, 'Emergence of new project teams from open source software developer networks: Impact of prior collaboration ties.', *Information Systems Research*, **19**(3), 369–391, (2008).

[13] Geir K. Hanssen, 'A longitudinal case study of an emerging software ecosystem: Implications for practice and theory', *J. Syst. Softw.*, **85**(7), 1455–1466, (July 2012).

[14] Qiaona Hong, Sunghun Kim, S. C. Cheung, and Christian Bird, 'Understanding a developer social network and its evolution', in *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pp. 323–332, Washington, DC, USA, (2011). IEEE Computer Society.

[15] S. Jansen, A. Finkelstein, and S. Brinkkemper, 'A sense of community: A research agenda for software ecosystems', in *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pp. 187–190, (May 2009).

[16] Aniket Kittur, Jeffrey V. Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton, 'The future of crowd work', in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pp. 1301–1318, New York, NY, USA, (2013). ACM.

[17] Theodoros Lappas, Kun Liu, and Evimaria Terzi, 'Finding a team of experts in social networks', in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pp. 467–476, New York, NY, USA, (2009). ACM.

[18] Anirban Majumder, Samik Datta, and K.V.M. Naidu, 'Capacitated team formation problem on social networks', in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pp. 1005–1013, New York, NY, USA, (2012). ACM.

[19] Konstantinos Manikas and Klaus Marius Hansen, 'Software ecosystems - a systematic literature review', *J. Syst. Softw.*, **86**(5), 1294–1306, (May 2013).

[20] Tom Mens, Malick Claes, Philippe Grosjean, and Alexander Serebrenik, 'Studying evolving software ecosystems based on ecological models', in *Evolving Software Systems*, eds., Tom Mens, Alexander Serebrenik, and Anthony Cleve, 297–326, Springer Berlin Heidelberg, (2014).

[21] Tom Mens and Philippe Grosjean, 'The ecology of software ecosystems', *IEEE Computer*, **48**(10), 85–87, (2015).

[22] David G. Messerschmitt and Clemens Szyperski, *Software Ecosystem: Understanding an Indispensable Technology and Industry*, MIT Press, Cambridge, MA, USA, 2003.

[23] Melanie Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, USA, 1998.

[24] Daniel Schall, 'Expertise ranking using activity and contextual link measures', *Data Knowl. Eng.*, **71**(1), 92–113, (2012).

[25] Daniel Schall, *Service Oriented Crowdsourcing: Architecture, Protocols and Algorithms*, Springer Briefs in Computer Science, Springer New York, New York, NY, USA, 2012.

[26] Daniel Schall, 'Formation and interaction patterns in social crowdsourcing environments', *Int. J. Commun. Netw. Distrib. Syst.*, **11**(1), 42–58, (June 2013).

[27] Daniel Schall, 'Measuring contextual partner importance in scientific collaboration networks', *Journal of Informetrics*, **7**(3), 730 – 736, (2013).

[28] Daniel Schall, *Social Network-Based Recommender Systems*, Springer International Publishing, 2015.

[29] M. Srinivas and L.M. Patnaik, 'Genetic algorithms: a survey', *Computer*, **27**(6), 17–26, (June 1994).

[30] Cédric Teyton, Marc Palyart, Jean-Rémy Falleri, Floréal Morandat, and Xavier Blanc, 'Automatic extraction of developer expertise', in *18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, pp. 8:1–8:10, New York, NY, USA, (2014). ACM.

[31] Hyeongon Wi, Seungjin Oh, Jungtae Mun, and Mooyoung Jung, 'A team formation model based on knowledge and collaboration', *Expert Systems with Applications*, **36**(5), 9121 – 9134, (2009).


[5] https://www.sfia-online.org