A Cognitive Attraction Network Approach to the Software Team Building Decision Problem

Omar Elshinnawey, Ghada Abuguyan, and Zohair Chentouf

Software Engineering Department, College of Information and Computer Sciences, King Saud University, Riyadh, KSA

zchentouf@ksu.edu.sa

Abstract. Software team selection can be done based on several criteria. One among them is the individual developers' productivity. However, the sole productivity cannot be accurate if one has to take into consideration the goals of the software project. Alignment of the productivity-based decision with the project’s goals is the aim of the present article. The proposed automatable solution is based on Cognitive Attraction Networks.

Keywords: Team Building; Software Engineering; Project Management; Cognitive Attraction Networks.

1. Introduction

The optimization of teamwork has been intensively studied in various areas [1-4]. The early interest in software teamwork first aimed at the structure of software teams [5-6] then researchers began to investigate on the software team performance factors, like in [7-9]. In [9], for instance, emphasis is put on the developers’ personality, the work environment, the structure of the team, leadership, and communication. The impact of the team performance on the software project’s goal achievement also has been studied. For example, [10-12] highlighted that impact in terms of productivity, product quality, and project success. Study of the emergence of individual developers’ factors at the team’s performance level was performed in [13-16].

The present investigation’s first objective is an automatic aid to software development team building based on individual developers’ performance. A second objective is to align the team member selection with the goals of the software project. The direct correlation between developer assignment and the software production success was highlighted in [17, 18]. In general, it is up to the project’s leaders to perform the assignment based on their experience of people, software constraints (e.g. reliability), and the project skill requirements. This task becomes cumbersome with medium and large size projects. That is because the number of possible combinations of the candidate software developers’ roles rapidly degenerates into a too large solution space. This makes it virtually impossible to put the hand on an optimal solution. Hence the need for a decision support system to help tackle the problem’s complexity. Besides the complexity of the problem, another research motivation comes from the fact that well reputed software process models like People - CMM [19], Personal Software Process – PSP [20], Team Software Process – TSP [8], and the Rational Unified Process – RUP [21], do not model the sub-process of developer assignment. Our research is also motivated by a third fact; there are only a few
related works to ours. Ngo-The and Ruhe\cite{22} proposed a method for assigning developers to software projects by decomposing a project to a series of releases. The assignment problem is solved for every release based on the developers’ competencies. However, the method is only applicable to mature software teams where the processes are fully defined, measured, and controlled. Tsai et al.\cite{23} proposed a method for selecting software developers which focuses on human and material resources rather than task scheduling. The method is meant to be used under dynamic and stochastic constraints. It considers two factors: the developer’s programming productivity and salary. Otero, et al.\cite{24} proposed a process to assign tasks to software developers while they do not completely fulfill the project required skills. The process takes into consideration the actual skills, the required expertise levels, and the priority of the tasks to perform.

At the best of our knowledge, there is no previous research work that addressed the problem of software task assignment under the constraint of fulfilling the project’s goals. This is the aim of our research. First, developers’ competences are mined from the bug-tracking system content related to previous projects. We then build a Cognitive Attraction Networks-based model which encompasses the developers’ competences and the project’s skill requirements and goals. The assignment algorithm is described along with a case study.

Section 2 of the present article suggests a method for mining developers’ competency from a bug-tracking system. Section 3 presents in detail the proposed task assignment algorithm. A case study is presented in Section 4. Section 5 concludes the work.

2. Mining Developers’ Competency

Developers’ competency needs to be known before assigning the new project’s tasks to them. We assume that all the candidate developers were involved in past software projects. The bug-tracking system should contain information about the programming and debugging tasks that belong to the past projects. We mine this pool of information to conclude the competency level of every developer. For this aim, we assume the following features that are present in popular bug-tracking systems, like\cite{25-26}:

- A bug-tracking system contains defect and task descriptions.
- Every defect or task was assigned to a single developer.
- Every defect or task had a start date, a desired end date, and an actual end date.

We consider the software developers’ competences listed in\cite{27-29}:

- Reasoning: the process of forming conclusions or inferences from facts or premises.
- Decision making: the ability to make the right decision on the right time.
- Judgment: to identify the right attributes of an existing software or human mechanism.
- Stress tolerance: the ability to cope up with work pressure.
- Openness to change: self-improvement with present and future software business needs.
- Teamwork and cooperation: the ability to interact with colleagues and cooperate with them.

Other competences need to be added as well:

- Problem solving: the ability to solve unexpected technical problems and manage possible risks.
- Required technical hands-on skills like Java, Linux, SQL, setting a small network for tests, etc.
- Duration estimation: the ability to deliver assigned tasks in time.

We assume that the project manager or any allowed team member has the responsibility of calculating the candidate developers’ competences. For every candidate $D_i$, the manager:

a) Selects a task $T_i$ that was performed by $D_i$ for some past projects.

b) Calculates the actual duration of the task as:

$$Actual\ Duration = \text{End Date} - \text{Start Date}$$

c) Determines the list of required competences and rates the difficulty degree of each on a scale of 1-5. For the reasoning competence, for example, this will be denoted $\text{required reasoning score}$.

d) Assesses the performance of the developer $D_j$ who was in charge of $T_i$ in
A Cognitive Attraction Network Approach to the Software Team Building Decision Problem

regard to every competence. For the reasoning skill, for instance, this will be denoted actual reasoning score, which is an integer number on the scale 1-5.

c) Calculates the performance factors of the developer Dj in regard to the task Ti. We adopt the following metrics, which is based on the above listed competences:

- Duration Deviation (DD):
  \[ DD = \frac{\text{actual duration} - \text{duration estimate}}{\text{duration estimate}} \]
- Reasoning Deviation (RD):
  \[ RD = \frac{\text{actual reasoning score} - \text{required reasoning score}}{\text{required reasoning score}} \]
- Decision Making Deviation (MD):
  \[ MD = \frac{\text{actual decision score} - \text{required decision score}}{\text{required decision score}} \]
- Judgment Deviation (JD):
  \[ JD = \frac{\text{actual judgment score} - \text{required judgment score}}{\text{required judgment score}} \]
- Stress tolerance Deviation (SD):
  \[ SD = \frac{\text{actual stress tolerance score} - \text{required stress tolerance score}}{\text{required stress tolerance score}} \]
- Openness to change Deviation (OD):
  \[ OD = \frac{\text{actual openness to change score} - \text{required openness to change score}}{\text{required openness to change score}} \]
- Team Work and cooperation Deviation (TWD):
  \[ TWD = \frac{\text{actual team work score} - \text{required team work score}}{\text{required team work score}} \]
- Problem solving Deviation (PD):
  \[ PD = \frac{\text{actual problem solving score} - \text{required problem solving score}}{\text{required problem solving score}} \]
- Technical Skill X Deviation (XD):
  \[ X_D = DD \frac{5}{\text{diff}} \]
  where: X \in \{\text{Java, PHP, ...}\}; 1 \leq \text{diff} \leq 5 is the difficulty rate of the required skill X.

f) Repeats steps a)-e) for every task that was performed by the developer Dj.

g) Calculates the competence gap of Dj with regard to every one of the above listed skills as the mean of that skill’s deviations. For example, the reasoning gap:

\[ RG = \text{Mean RD} \]

where Mean RD is the mean of RD of all the tasks performed by the developer Dj. Similarly, all the other competence gaps will be calculated, namely: Duration Gap (DG), Decision Making Gap (MG), Judgment Gap (JG), Stress tolerance Gap (SG), Openness to change Gap (OG), Team Work and cooperation Gap (TWG), Problem solving Gap (PG), and Technical Skill X Gap (XG).

3. Software Task Assignment Decision Problem and Solution

Recall the problem being tackled by the present research. Given a new software development project, the related list of software development tasks, the list of competences required for every task, the list of candidate developers along with their competence levels mined from past projects’ data (see Section 2), and the list of goals to fulfill by the project, the problem to solve is one of assigning the best candidate developer to every development task in a way that maximizes the goal satisfaction and takes into consideration the competence requirements of the task and the corresponding actual competence levels of the developers. Note that a goal can be a non-functional requirement, such as reliability, efficiency, usability, availability, portability, maintainability, etc., or a project constraint such as time or budget. Formally, a software task assignment decision problem constituents are: a set of tasks Ti, i=1, …, NT, a set of goals Gk, k=1, …, NG, and a set of developers Dj, j=1, …, ND. Every goal Gk has an importance weight pk \in [0, 1], with \( \sum_{k=1}^{NG} p_k = 1 \). Every task Ti contributes with a weight \( w_{ki} \in [0, 1] \) in satisfying the goal Gk, with \( \sum_{i=1}^{NT} w_{ki} = 1 \). Every task Ti requires a competence a set of competences Cu, u=1, …, NC and a set of developers Dj, j=1, …, ND. Every goal Gk has an importance level \( r_{iu} \in [0, 1] \), \( \sum_{u=1}^{NC} r_{iu} = 1 \). The competence gap of a developer Dj with regard to a competence Cu will be denoted \( c_{guj} \in [-0.8, 4] \). The latter interval values are engendered by the fact that skill deviations in Section 2 were based on a 1-5 scale.
We use these formulation elements to construct a Cognitive Attraction Network (CAN). CAN was introduced in [30-31]. Figure 1 depicts the developer assignment problem’s CAN with a single task. The assignment problem can be formulated as selecting a developer $D_j$ among the set of $ND$ candidate developers to be in charge of a given task $T_i$ under the constraint of fulfilling the goals $G_1, \ldots, G_{NG}$. For this aim, we need to calculate the so-called cognitive attraction [30] of the goals made on developers through the following steps:

a) The attraction of the task $T_i$ performed on every developer $D_j$:

$$ t_{ij} = \sum_{u=1}^{NC} r_{iu} * c g_{uj} $$

b) The attraction of every goal $G_k$ performed on every developer $D_j$:

$$ g_{kj} = p_k * w_{ki} * t_{ij} $$

c) The sum of goal attractions made on the developer $D_j$:

$$ g_j = \sum_{k=1}^{NG} g_{kj} $$

Since $g_j$ is a function of the developer's skill gaps $c_{g_{uj}}$, the developer with the smallest value of $g_j$ is selected for the task $T_i$.

d) Repeat steps a)-c) for every other task $T_v$ excluding all the developers $D_i$ who were selected for previous tasks.

**4. Case Study**

Let us take an illustrative example. Suppose that we have three developers: $D_1$, $D_2$, and $D_3$, one task $T_i$, three required competences $C_1$, $C_2$, and $C_3$, and four goals $G_1$, $G_2$, $G_3$ and $G_4$. The assignment algorithm inputs and processing results are summarized in Table 1-Table 6.

**Table 1. Importance Weights of Goals ($p_k$)**

<table>
<thead>
<tr>
<th></th>
<th>$G_1$</th>
<th>$G_2$</th>
<th>$G_3$</th>
<th>$G_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.150</td>
<td>0.150</td>
<td>0.300</td>
<td>0.400</td>
</tr>
</tbody>
</table>

**Table 2. Contributions of the Task $T_i$ in Goals ($w_{ki}$)**

<table>
<thead>
<tr>
<th></th>
<th>$G_1$</th>
<th>$G_2$</th>
<th>$G_3$</th>
<th>$G_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>0.300</td>
<td>0.400</td>
<td>0.200</td>
<td>0.500</td>
</tr>
</tbody>
</table>

**Table 3. Competences’ Importance Weights for the Task $T_i$ ($r_{iu}$)**

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$C_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>0.200</td>
<td>0.300</td>
<td>0.500</td>
</tr>
</tbody>
</table>

**Table 4. Developers’ Competence Gaps ($c_{g_{uj}}$)**

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>-0.200</td>
<td>1.500</td>
<td>-0.750</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.250</td>
<td>-0.500</td>
<td>0.670</td>
</tr>
<tr>
<td>$C_3$</td>
<td>0.670</td>
<td>0.250</td>
<td>0.670</td>
</tr>
</tbody>
</table>

**Table 5. Attraction of $T_i$ on Developers ($t_{ij}$)**

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i$</td>
<td>0.370</td>
<td>0.275</td>
<td>0.386</td>
</tr>
</tbody>
</table>

**Table 6. Attraction of Goals on Developers ($g_{kj}$ and $g_j$)**

<table>
<thead>
<tr>
<th></th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>0.017</td>
<td>0.012</td>
<td>0.017</td>
</tr>
<tr>
<td>$G_2$</td>
<td>0.022</td>
<td>0.017</td>
<td>0.023</td>
</tr>
<tr>
<td>$G_3$</td>
<td>0.022</td>
<td>0.017</td>
<td>0.023</td>
</tr>
<tr>
<td>$G_4$</td>
<td>0.074</td>
<td>0.055</td>
<td>0.077</td>
</tr>
<tr>
<td>Sum</td>
<td>0.135</td>
<td>0.100</td>
<td>0.141</td>
</tr>
</tbody>
</table>
Based on the results of Table 6, the best candidate for the task $T_1$ is the developer $D_2$.

5. Conclusion

This paper presented a Cognitive Attraction Network-based algorithm destined to support the project manager in team selection under the constraints of software project’s goals, skill requirements, and candidate developers’ actual competences. These competences need to be mined from the bug-tracking system of the company. Specifically, the data related to the past tasks completed by the candidate developers are to be exploited to infer their individual competence gaps. The proposed mining method can easily be automated as a feature of the bug-tracking system itself. In this way, the task of calculating the developers’ competence gaps would be dramatically simplified. The proposed task assignment algorithm builds on the results of this competency calculation. The algorithm’s output is a ranking of developers’ suitability for every project’s task.

As future work, there should be an empirical validation through real utilization and evaluation of the proposed solution.

References


مقارية لتشكّيل فرق مبرمجين باستخدام شبكات الجذب المعرفي

عمر الشناوي، وغادة أبوقيان، وزيّر شن توف

قسم هندسة البرمجيات، كلية علوم الحاسب والمعلومات، جامعة الملك سعود

الرياض، المملكة العربية السعودية

المتسلسل. تعتمد عملية اختيار أعضاء فريق إنجاز مشروع برمجيات على كثير من المعايير من بينها إنتاجية المبرمجين. لكن الاعتماد على الإنتاجية وحدها يجانب الدقة في كثير من الأحيان، خصوصًا عند النظر إلى علاقتها بأهداف المشروع ككلة. أول هدف لهذه الورقة البحثية هو إيجاد نموذج مفهومي يربط بين القرارات المبنية على الإنتاجية وأهداف المشروع على اختلافها. ثاني الهدف هو تأثيم النموذج، باستعمال شبكات الجذب المعرفي.