A Cognitive System to Teach Software Maintenance Project Staffing

Zohair Chentouf [1]

[1] *College of Computer and Information Sciences, KS University, Riyadh, KSA*

Abstract **–This paper describes a cognitive tool for teaching maintenance software project staffing decision making, which is based on a computational model of the human staffing decision process. The learning activity requires students to solve the given maintenance staffing problem before running the tool and comparing their decision with the tool's one.**

Keywords **– Software project staffing, Project-based learning, Competition-based learning, Decision making.**

1. Introduction

Project Management is known to be a supporting discipline to the software process. That is why the IEEE SWEBOK, for example, recommends that Project Management be part of software engineering curricula. The targeted educational outcomes consist of learning necessary technical knowledge, as well as acquiring non-technical skills such as autonomy and pro-activity that allow students to deal with real world situations [1]. Because of such an integration of technical and non-technical skills, however, software engineering teaching is becoming more complex [2].

DOI: 10.18421/TEM64-08
https://dx.doi.org/10.18421/TEM64-08

Corresponding author: Zohair Chentouf,
College of Computer and Information Sciences, KS University, Riyadh, KSA
Email: zchentouf@ksu.edu.sa

Received: 14 September 2017
Accepted: 20 October 2017
Published: 27 November 2017

Management of software maintenance projects is a typical software management activity where both technical and non-technical skills of developers must be considered. Software maintenance is the process of changing software after its deployment. Changes can consist of fixing defects (corrective maintenance), adapting the software to an evolving environment (adaptive maintenance), adding new features (perfective maintenance), or preventing defects (preventive maintenance). The study performed in the USA by Glass [3] revealed that software maintenance projects consume up to 80% of development costs. Jones [4] conducted another investigation on American software companies; he reported that most of small-medium companies and part of big companies attribute maintenance tasks to developers who are already involved in development projects. He also pointed out that in many cases a developer has to maintain an application that (s)he did not develop, coded in a very old language, and with no written specification [4]. In these work conditions, communication is slow and disruptive conflicts among developers are not uncommon [4]. All these software maintenance projects' characteristics make staffing such projects a complex task [3],[4].

The here reported learning activity aimed to allow students apprehend the problem within all its aspects. It was designed as staffing decision problem resolution, where students need to choose among two candidate developers under the constraints of the maintenance project, the development projects in which they are already involved, and their technical and non-technical skills.

Most of the university software project management courses are based on the Project Management body of knowledge and address scheduling, planning, and quality control. To deal with specific software engineering aspects, they focus on software implementation [2],[5]. For practice and assignments, those courses usually use MS-Project [5]. Unfortunately, the so designed software project management courses do not address the software maintenance project staffing problem. Furthermore, students usually find learning MS-Project a cumbersome assignment [6].

Our research was motivated by the lack of specific software maintenance project staffing tools and the complexity of this software management activity. The complexity of this activity is a typical indicator of the reasons behind the reputation of software engineering education for producing insufficiently prepared graduates [7]. In order to achieve satisfactory learning results despite the complexity of the problem, the learning activity was conducted as a practical one, using software project case studies, under a combined Project Based Learning (PBL) and competition approach.

Nowadays PBL is accepted as one of the best teaching practices [8-9]. It has been validated as a successful approach to enhancing young software engineers leadership and teamwork skills [8,10]. The PBL learning process first introduces students to the problem aspects then requires them to jointly elaborate a solution [11]. According to [8],[12],[13], a typical PBL addresses open-ended and challenging problems which have a variety of contexts. A successful PBL activity is one that allows students to work within teams, infer new knowledge, apply it to solve the problem, and identify the weaknesses of their own methods. In general, PBL assigns the instructor a role of learning facilitator. As per these educational properties, PBL is becoming popular in higher education [14]. Its success in software engineering was validated in [15], for example.

PBL was applied to teaching software project management, particularly to enhancing collaboration skills among software engineering students through a collaborative network [16],[17] and integrating project management and software processes [18]. However, maintenance software project staffing was not addressed.

Competition is a useful teaching instrument that aims to stimulate students' learning motivation [19]. Culha [20] used the competition-based method in software engineering education. The objective was to teach students the principles of the agile software process, namely self-motivation, development speed, and adaptation to changing requirements. At the best of our knowledge, there is no work that used competition-based learning to teach software maintenance project staffing.

Section 2 presents the learning objectives of the presented activity. Section 3 describes the maintenance project staffing problem, the proposed solution model, and the tool's implementation details. Section 4 describes the process of designing and conducting the learning activity. Section 5 presents experimental results based on learners' feedback. Section 6 concludes the work and suggests some future direction.

2. Objectives

The here described learning activity was introduced in a software process management course at the last year of MS degree in Software Engineering. Prior to this course, students were exposed to advanced software engineering topics related to software requirement engineering, software design and architecting, and software project management.

To fill the gap of software maintenance project staffing, the learning activity aimed at reinforcing relevant cognitive skills, namely: analysis, synthesis, and communication. The relationships between these cognitive skills and the course learning outcomes that correspond to maintenance software project staffing are in Table 1. It is worth noting that the course has other learning outcomes which do not map to the staffing problem.

Table 1. Course learning outcomes and cognitive skills

Course learning outcome	Cognitive skill
1. Estimating developer productivity based on technical skills	Analyzing new situations
2. Estimating developer productivity based on non-technical skills	Analyzing new situations
3. Evaluating the importance of project parameters in staffing decision	Analyzing new situations
4. Evaluating the impact of skills and project parameters on each other	Synthesizing
5. Managing project's people	Communicating and convincing others

3. Maintenance project staffing decision modeling and implementation

The system that was used in the learning activity was based on a model that is a synthesis of [21-26]. The staffing decision model considers two candidate developers V_1 and V_2, everyone having a set of technical skills T_i, i=1,…, 13, a set of non-technical skills N_j, j=1, …, 4, and a set of project management constraints C_k, k=1, 2; T_i, N_j, $C_k \in [0, 1]$.

Technical skills

- Architectural design (T_1)
- Low level design (T_2)
- Data structure design (T_3)
- Algorithm design (T_4)
- Interface design (T_5)
- Graphical design (T_6)
- Coding and unit testing (T_7)
- Other testing techniques (T_8)
- Integration testing (T_9)
- System testing (T_{10})
- Performance testing (T_{11})
- Acceptance testing (T_{12})
- Familiarity (T_{13})

All the technical skill values T_i measure the developer's skill degree. For example, T_3=0.4 indicates that the developer's proficiency in data structure design is estimated at 40%. Note that "other testing" (T_8) includes all the testing techniques that are not listed here, and "familiarity" (T_{13}) corresponds to the extent to which the candidate developer knows the code to maintain.

Non-technical skills

- Reasoning and judgment (N_1)
- Decision and problem solving (N_2)
- Stress tolerance (N_3)
- Openness, communication and team work (N_4)

These technical and non-technical skill variables are assumed to have already been measured by the project manager through past projects.

PM constraints

- Employee low cost (C_1)
- Context change low cost (C_2)

To make it clearer, the greater C_1 the less costly is the candidate developer; the greater C_2, the less negative impact of having the developer leave his/ her current duties in other projects to work on the maintenance one.

Other decision parameters represent the project's requirements and constraints: project characteristics P_i, i=1, 2, maintenance scope parameters M_j, j=1, …, 4, degree of the code change depth D_k, k=1,2, and product quality values Q_l, l=1,2; P_i, M_j, D_k, $Q_l \in [0, 1]$.

Project characteristics

- Project and/ or customer importance (P_1)
- Task criticality (P_2)

The project/ customer importance may be strategic, competitive, financial, etc. The task criticality should be seen as reflecting the criticality of the software to maintain.

Maintenance scope

- Correction (M_1)
- Adaption (M_2)
- Perfection (M_3)
- Prevention (M_4)

A software maintenance project may include all or part of these maintenance types.

Change depth

- New low level design (D_1)
- New technology (D_2)

The former parameter estimates the extent of changes on the low level design, and the latter reflects how much new technology will be introduced.

Product quality

- Bad documentation (Q_1)
- Bad code quality (Q_2)

These parameters estimate how poor are the specification documentation and the source code, respectively. In other words, the greater Q_1, the less good documentation of the software to maintain; the greater Q_2, the less good code to change.

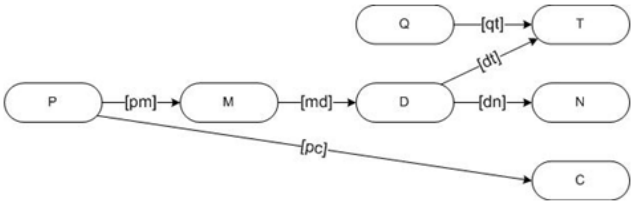

Figure 1. Maintenance project staffing CAN

The previously presented parameters were embedded on a Cognitive Attraction Network (CAN) to allow inference of a decision. CAN were used in [27]. Roughly, CAN is a directed graph of concept layers. Two adjacent layers L_1 and L_2 model the fact that any change of the status of L_1 elements impacts (causes the change of) those of L_2. The status of conceptual elements and the impact relationships that exist between them are quantified using numerical values.

The CAN depicted by Fig. 1 models our problem of software maintenance project staffing and shows the impact relationships that exists between parameter groups: P represents project parameters P_i, M represents maintenance scope parameters M_j, etc. The parameters of any two adjacent layers are fully connected by impact links. For example, every Pi,

i=1,2, has an impact link towards every Mj, j=1,4. Impact links are assigned values in [0,1]. Hence the impact matrices pm, md, etc. (see Fig. 1).

A decision inference process is based on impact propagation from P to M, M to D, D and Q to T, D to N, and P to C. First, the impact of P_i causes updating the values M_j, which are calculated using P_i, M_j, and pm_{ij}. Similarly, the impact of the maintenance type parameters M_j propagates through the matrix md to update the change depth parameters D_k. This process continues until T_i, N_j, and C_k are updated, which corresponds to a computation of the skills and project constraints of the candidate developer. To calculate T_i, N_j, and C_k of the second candidate developer, the same process has to be run. The developer to select will be the one with the greater sum of T_i, N_j, and C_k. The matrices pm, md, qt, dt, dn, and pc are assumed to have been refined by the PM through past software maintenance projects.

Staffing decision algorithm

The whole decision problem algorithm can be formulated as:

i) Update maintenance scope values:
For j=1 to 4:
$$M_j = M_j * \sum_{i=1}^{2} pm_{ij} * P_i \quad (1)$$

ii) Update change depth values:
For j=1 to 2:
$$D_j = D_j * \sum_{i=1}^{4} md_{ij} * M_i \quad (2)$$

iii) Update PM constraints values of the developers V_k:
For k=1 to 2:
For j=1 to 2:
$$C_j^k = C_j^k * \sum_{i=1}^{2} pc_{ij} * P_i \quad (3)$$

iv) Update technical skills values of the developers V_k:
For k=1 to 2:
For j=1 to 13:
$$T_j^k = T_j^k * [\sum_{i=1}^{2} qt_{ij} * Q_i + \sum_{p=1}^{2} dt_{pj} * D_p] \quad (4)$$

v) Update non-technical skills values of the developers V_k:
For k=1 to 2:
For j=1 to 4:
$$N_j^k = N_j^k * \sum_{i=1}^{2} dn_{ij} * D_i \quad (5)$$

vi) Calculate the total score of every developer V_k:
For k=1 to 2:
$$S^k = \sum_{i=1}^{13} T_i^k + \sum_{j=1}^{4} N_j^k + \sum_{p=1}^{2} C_p^k \quad (6)$$

vii) Select developer V_k who has:
$$S^k = \max(S^1, S^2) \quad (7)$$

Example

Fig. 2 depicts part of a sample maintenance project staffing CAN for which the project manager considers that the current project to maintain is important at 60% (P_1=0.6), 40% (M_2=0.4) of it corresponds to correction, 60% (D_2=0.6) implies new technologies, and 60% (Q_1=0.6) of the documentation is inexistent or poorly written. The sample CAN also shows the coding (T_7) and reasoning (N_1) skill levels of the two candidate developers, as well as their context change parameter (C_2). The arrows are labeled with impact weights; for example, pm_{12}=0.7, indicates that through the past projects, the project manager concluded that the maintenance project importance (P_1) impacts by 70% the value of correction maintenance (M_2).

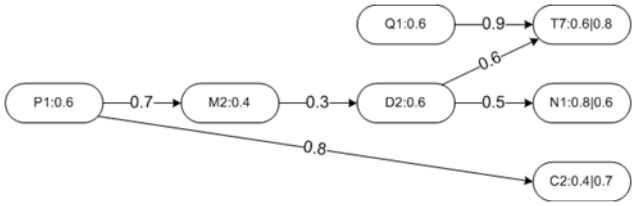

Figure 2. Sample maintenance project staffing CAN

Let us apply the decision algorithm:

i) Update maintenance scope values:
$M_2 = M_2 * pm_{12} * P_1 = 0.4 * 0.7 * 0.6 = 0.168$

ii) Update change depth values:
$D_2 = D_2 * md_{22} * M_2$
$= 0.6 * 0.3 * 0.168 = 0.030$

iii) Update PM constraint values of the developer V_1:
$C_2^1 = C_2^1 * pc_{12} * P_1$
$= 0.4 * 0.8 * 0.6 = 0.192$
Update PM constraint values of the developer V_2:
$C_2^2 = C_2^2 * pc_{12} * P_1$
$= 0.7 * 0.8 * 0.6 = 0.336$

iv) Update technical skill values of the developer V_1:
$T_7^1 = T_7^1 * [qt_{17} * Q_1 + dt_{27} * D_2]$
$= 0.6 * [0.9 * 0.6 + 0.6 * 0.030] = 0.335$
Update technical skill values of the developer V_2:
$T_7^2 = T_7^2 * [qt_{17} * Q_1 + dt_{27} * D_2]$
$= 0.8 * [0.9 * 0.6 + 0.6 * 0.030] = 0.446$

v) Update non-technical skill values of the developer V_1:
$N_1^1 = N_1^1 * dn_{21} * D_2$
$= 0.8 * 0.5 * 0.030 = 0.012$
Update non-technical skill values of the developer V_2:

$N_1^2 = N_1^2 * dn_{21} * D_2$
$= 0.6 * 0.5 * 0.03 = 0.009$

vi) Calculate the total score of the two developers:
$S^1 = T_7^1 + N_1^1 + C_2^1 = 0.335 + 0.012 + 0.192 = 0.539$
$S^2 = T_7^2 + N_1^2 + C_2^2 = 0.446 + 0.009 + 0.336 = 0.791$

vii) Select developer V_k who has:
$S^k = \max(S^1, S^2)$
$= \max(0.539, 0.791) = 0.791$
Developer V_2 is then selected.

Impact weights

As stated above, the PM is supposed to have elaborated the matrices pm, md, qt, dt, dn, and pc through past maintenance projects. For the purpose of our learning activity, the instructor and students determined the values of those matrices (see Table 2 – Table 7). Note that students were introduced to the problem as formulated in this article but for usability reasons, the implemented system contained 9 technical skills out of 13. That is why Table 4 contains 9 lines only.

Table 2. Values of the matrix pm

	Correction (M_1)	Adaption (M_2)	Perfection (M_3)	Prevention (M_4)
Project Importance (P_1)	0.5	0.35	0.6	0.35
Task criticality (P_2)	0.5	0.65	0.4	0.65

Table 3. Values of the matrix md

	New low design (D_1)	New Technology (D_2)
Correction (M_1)	0.25	0.0
Adaption (M_2)	0.35	0.6
Perfection (M_3)	0.2	0.15
Prevention (M_4)	0.2	0.25

Table 4. Values of the matrix qt

	Bad Doc. (Q_1)	Bad Code (Q_2)
Arch. Des. (T1)	0.8	0.2
L.L. Des. (T2)	0.6	0.4
D.S. Des. (T3)	0.4	0.6
Algo. Des. (T4)	0.3	0.7
Cod. &UT (T7)	0.2	0.8
Oth. Test. (T8)	0.5	0.5
Integ. Test. (T9)	0.5	0.5
Sys. Test. (T10)	0.7	0.3
Fam. (T13)	0.3	0.7

Table 5. Values of the matrix dt

	New low design (D1)	New Technology (D2)
Arch. Des. (T1)	0.2	0.8
L.L. Des. (T2)	0.8	0.2
D.S. Des. (T3)	0.8	0.2
Algo. Des. (T4)	0.8	0.2
Cod. &UT (T7)	0.6	0.4
Oth. Test. (T8)	0.35	0.65
Integ. Test. (T9)	0.3	0.7
Sys. Test. (T10)	0.3	0.7
Fam. (T13)	0.7	0.3

Table 6. Values of the matrix dn

	New low design (D_1)	New Technology (D_2)
Reasoning (N1)	0.3	0.7
Decision making (N2)	0.3	0.7
Stress tolerance (N3)	0.3	0.7
Communic. (N4)	0.4	0.6

Table 7. Values of the matrix pc

	Employee cost (C1)	Context change cost (C2)
Project Importance (P_1)	0.4	0.6
Task criticality (P_2)	0.35	0.65

System implementation and handling

The above described decision algorithm was implemented in Java as the core of a cognitive agent. Decision parameters need to be written in text file. The agent loads the parameters on the GUI. Then, the user pushes a button to order the agent to calculate a decision. The result is displayed on the GUI. Fig. 3 shows a sample GUI screen.

As stated above, a decision problem is based on the comparison between two candidate developers. To decide which developer to select, student groups need to compare their skill levels. As depicted by Fig. 3, numerical values of skill levels were replaced by a graphical notation. For example, Developer V_2 is better than Developer V_1 in algorithm design (T_4) and the difference of skill levels is equal to one unit; a unit being equal to [0.1,0.4[. This can be expressed as: $0.1 \leq T_4^2 - T_4^1 < 0.4$. This implies that two star difference, like in architecture design (T_1), means: $0.4 \leq T_1^2 - T_1^1 < 0.7$. If $0.7 \leq T_i^2 - T_i^1 \leq 1.0$, the difference in skill T_i level will be displayed with three stars. When $0.0 \leq T_1^2 - T_2^1 < 0.1$ the skill levels of the two developers are considered equal and thus no one is assigned a star; both are assigned a hyphen. The figure on the bottom of the screen helps students recall the impact relationships between the different decision parameters. When the students have made a decision, the instructor clicks the button "Compute" and the decision result is then displayed on screen. On Fig. 3 developers V_1 and V_2 scored 2.4 and 2.1, respectively.

During the first exercise sessions, students tended to select the developer who has more stars. Through the exercise sessions, students figured out that the impact level to which are subject the skill parameters must be taken into consideration. They then asked to see the value of that impact. Figure 4 shows an excerpt of the system's log. Students finally came up with a decision process in which they examine the parameters on the GUI and the impact values on the log file. Their decision became better compared with the first sessions.

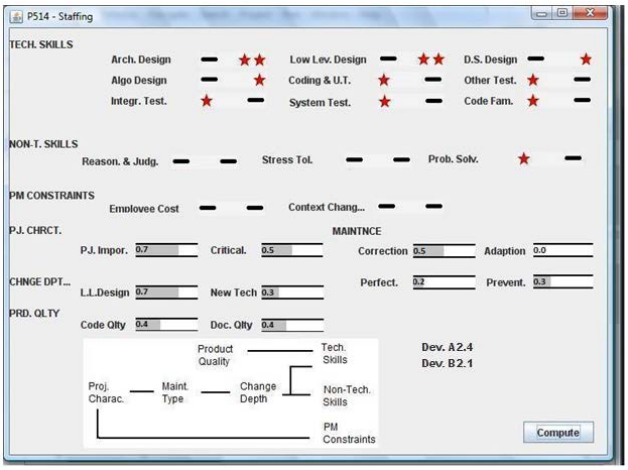

Figure 3. Screen of the software maintenance staffing decision tool

```
Average Impact of Change Depth on N-T Skills 0.595
Average Impact of Prod. Q. Change Dep. on T. Skills 0.26444447
Average Impact of Proj. Chara. on Proj. Constr. 0.45
```

Figure 4. Excerpt of the software maintenance staffing decision tool's log file

4. Learning activity design and process

The learning activity was introduced as the basis for a tutorial session of three hours. Prior to that, students were introduced to theoretical aspects of the maintenance project staffing problem during one lecture session. The students were guided by the instructor to elaborate the values of the matrices pm, md, qt, dt, dn, and pc. As stated earlier, the activity was designed as a project and competition-based one. As such, it relies on students' feedback to guide their own learning. The class contains 20 students who work within five groups of four students. Groups are assigned the task to solve 5 different cases. The detailed description of parameters of the cases is given to students one week earlier to thoroughly study. Student groups compete in class to make a staffing decision that coincides with the system's inference. First, every case study is presented in detail by the instructor using the system's interface. After having answered the students' questions about the case study, student groups are given 10 minutes to debate and conclude a decision. The instructor then runs the system to infer a decision, and finally every group is given 3 minutes to explain their collective reasoning that led to their decision. A free discussion of 15 minutes allows students to give their feedback about their own decision process and on the other groups' reasoning. During discussion the instructor plays the role of moderator.

5. Activity and tool validation

The proposed learning activity was experimented with MSc students (KS University, second semester 2016–2017), within the Software Processes and Construction course. To assess the achievement of the learning outcomes students' feedback was surveyed. Table 8 contains the questionnaire and the responses of 20 students.

- A.1–A.5 cover the course learning outcomes and related cognitive skill improvement. According to the results, most of the students (65%-95%) feel they reached those outcomes and ameliorated the corresponding cognitive skills.
- B relates to the student's confidence in own ability to benefit from this learning in real professional life. 80%-90% of students are satisfied with this aspect.
- C is about the usability of the system. 55%-80% of students were satisfied with the system's usability, which is a satisfactory result.
- D assesses the usefulness of the system. Students' satisfaction was high, 70%-95%.

Overall, students' feedback was positive, which let us conclude that the learning activity was successful and thus we are confident of its potential.

6. Conclusion

This article reported a teaching activity that is part of a software process management course at the last year of MS degree in Software Engineering. The main goal was to fill the current gap in software engineering education that relates to maintenance software project staffing decision making, which is an important and complex project management task. To cope with this problem a decision model was designed as a Cognitive Attraction Network (CAN) and embedded on a cognitive agent-based tool. The conducted teaching experiment proved the efficiency of the tool as a learning assistant and a means to reach the course learning outcomes.

As future work, the staffing decision model should be integrated as part of a software project management tools package to automate the system's inputs.

Acknowledgements

This work has been supported by the Research Center of the College of Computer and Information Sciences at King Saud University. The author is grateful for this support.

Table 8. Student inquiry questions and results

5: maximum score; 1: minimum score

Questions		5 [%]	4 [%]	3 [%]	2 [%]	1 [%]	5+4 [%]
A	I feel that my skills of X have improved in regard to Y						
A.1	X: Analyzing new situations Y: Estimating developer productivity based on technical skills	13 [65]	6 [30]	1 [5]	0 [0]	0 [0]	19 [95]
A.2	X: Analyzing new situations Y: Estimating developer productivity based on non-technical skills	16 [80]	2 [10]	2 [10]	0 [0]	0 [0]	18 [90]
A.3	X: Analyzing new situations Y: Evaluating the importance of project parameters in staffing decision	12 [60]	4 [20]	3 [15]	1 [5]	0 [0]	16 [80]
A.4	X: Synthesizing all the decision problem parameters Y: Staffing decision	15 [75]	2 [10]	3 [15]	0 [0]	0 [0]	17 [85]
A.5	X: Communicating, convincing others Y: Managing project's people	19 [95]	1 [5]	0 [0]	0 [0]	0 [0]	20 [100]
B	I feel that I am ready to positively contribute to solve the software maintenance staffing decision problem in my job environment.	13 [65]	5 [25]	1 [5]	1 [5]	0 [0]	18 [90]
C	The configuration and usage of the system is easy.	11 [55]	5 [25]	2 [10]	2 [10]	0 [0]	16 [80]
D	The system is important and useful.	14 [70]	5 [25]	0 [0]	1 [5]	0 [0]	19 [95]

References

[1] Salas-Morera, L., Arauzo-Azofra, A., García-Hernández, L., Palomo-Romero, J. M., & Hervás-Martínez, C. (2013). PpcProject: An educational tool for software project management. *Computers & Education*, *69*, 181-188.

[2] González-Morales, D., de Antonio, L. M. M., & García, J. L. R. (2011, April). Teaching "Soft" skills in software engineering. In Global Engineering Education Conference (EDUCON), 2011 IEEE (pp. 630-637). IEEE.

[3] R. L. Glass, *Facts and Fallacies of Software Engineering*, Addison Wesley, 2002.

[4] C. Jones, *Software Engineering Best Practices*, McGraw-Hill, 2010.

[5] Ivanovic, M., Putnik, Z., Budimac, Z., & Bothe, K. (2012, April). Teaching "Software Project Management" course-seven years experience. In Global Engineering Education Conference (EDUCON), 2012 IEEE (pp. 1-7). IEEE.

[6] Tatnall, A., & Reyes, G. (2005). Teaching IT Project Management to Postgraduate Business Students: A Practical Approach. *Journal of Information Technology Education*, *4*., 153–166.

[7] Hainey, T., Connolly, T. M., Stansfield, M., & Boyle, E. A. (2011). Evaluation of a game to teach requirements collection and analysis in software engineering at tertiary education level. *Computers & Education*, *56*(1), 21-35.

[8] I.A. Garcia and C.L. Pacheco.(2014). Using TSPi and PBL to Support Software Engineering Education in an Upper-Level Undergraduate Course, *Computer Applications in Engineering Education* 22, 736–749.

[9] Ghezzi, C., & Mandrioli, D. (2005, May). The challenges of software engineering education. In *Proceedings of the 27th international conference on Software engineering* (pp. 637-638). ACM.

[10] Mead, N. R. (2009). Software engineering education: How far we've come and how far we have to go. *Journal of Systems and Software*, *82*(4), 571-575.

[11] Anisetty, P., & Young, P. (2011). Collaboration problems in conducting a group project in a software engineering course. *Journal of Computing Sciences in Colleges*, *26*(5), 45-52.

[12] Brodie, L., Zhou, H., & Gibbons, A. (2008). Steps in developing an advanced software engineering course using problem based learning. *engineering education*, *3*(1), 2-12.

[13] Gibbings, P., & Brodie, L. (2008). Team-based learning communities in virtual space. *International Journal of Engineering Education*, *24*(6), 1119.

[14] P. Dillenbourg, What do you mean by collaborative learning?, In: *Collaborative-learning: Cognitive and computational approaches*, P. Dillenbourg (Ed.), Elsevier, Oxford, 1999.

[15] I. Richardson, L. Reid, S.B. Seidman, B. Pattinson, and Y. Delaney, Educating software engineers of the future: Software quality research through problem-based learning, *24th IEEE CS Conference on Software Engineering Education and Training*, IEEE Computer Society, 2011, pp. 91–100.

[16] Garcia, I. A., Calvo-Manzano, J. A., Pacheco, C. L., & Perez, C. A. (2015). Software engineering education for a graduate course: A web-based tool for conducting process improvement initiatives with local industry collaboration. *Computer Applications in Engineering Education*, *23*(1), 117-136.

[17] Gregoriou, G., Kirytopoulos, K., & Kiriklidis, C. (2013). Project management educational software (ProMES). *Computer Applications in Engineering Education*, *21*(1), 46-59.

[18] Martínez, L. G., Licea, G., Juárez, J. R., & Aguilar, L. (2014). Experiences using PSP and XP to support teaching in undergraduate programming courses. *Computer Applications in Engineering Education*, *22*(3), 563-569.

[19] Altin, H., & Pedaste, M. (2013). Learning approaches to applying robotics in science education. *Journal of baltic science education*, *12*(3), 365-377.

[20] Çulha, D. (2016). Applying competition-based learning to agile software engineering. *Computer Applications in Engineering Education*, *24*(3), 382-387.

[21] Crawford, B., Soto, R., Johnson, F., Monfroy, E., & Paredes, F. (2014). A max–min ant system algorithm to solve the software project scheduling problem. *Expert Systems with Applications*, *41*(15), 6634-6645.

[22] Xiao, J., Ao, X. T., & Tang, Y. (2013). Solving software project scheduling problems with ant colony optimization. *Computers & Operations Research*, *40*(1), 33-46.

[23] Yannibelli, V., & Amandi, A. (2011). A knowledge-based evolutionary assistant to software development project scheduling. *Expert Systems with Applications*, *38*(7), 8403-8413.

[24] Feng, B., Jiang, Z. Z., Fan, Z. P., & Fu, N. (2010). A method for member selection of cross-functional teams using the individual and collaborative performances. *European Journal of Operational Research*, *203*(3), 652-661.

[25] Tanrıöver, Ö. Ö., & Demirörs, O. (2015). A process capability based assessment model for software workforce in emergent software organizations. *Computer Standards & Interfaces*, *37*, 29-40.

[26] da Silva, F. Q., Franca, A. C. C., Gouveia, T. B., Monteiro, C. V., Cardozo, E. S., & Suassuna, M. (2011, September). An empirical study on the use of team building criteria in software projects. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (pp. 58-67). IEEE.

[27] Chentouf, Z. (2016). Design and evaluation of a cognitive system to teach software change control. *Computer Applications in Engineering Education*, *24*(3), 347-355.