Learning outcomes in a project-oriented agile environment

Hadas Chassidim†, Guy Leshem‡, Dov Sohacheski†, Dani Almog† & Shlomo Mark†

Shamoon College of Engineering, Ashdod, Israel†
Ashkelon Academic College, Ashkelon, Israel‡

ABSTRACT: Despite the vast implementation of project-oriented courses in advanced academic curricula, little is known about their efficiency in properly transmitting material to students. The idea behind this research is to identify the advantages and disadvantages of learning outcomes in a project-oriented environment. A project-oriented course, spanning a single semester across two campuses, was used as a method of measuring various learning outcomes. Students partook in the research both explicitly and implicitly. Staff members' continuous inquiry into the pace and success of the students' advancement of the project, together with the direct feedback from the students, yields a substantial subset of the overall results. Ultimately, a tailored survey, consisting of questions regarding the original learning outcomes was statistically analysed to verify the researchers' observations throughout the duration of the course. Initial results imply that the course significantly contributed to the student's professional expertise in the field.

INTRODUCTION

The profession of software engineering is unique and dynamic, and requires combining skills from different fields. The training should follow the academic requirements, as well as prepare students for future employment as software engineers. The willingness of a student to learn, together with the desire of an instructor to teach is not enough in the field of agile [1-2].

Communication, collaboration, constant change and other agile considerations cannot be fully mimicked in the classroom environment. In practice, developers can have progressive communication and successful collaboration with their team members, the Scrum master and even stakeholders on a certain project, yet display complete deficiency on another. Properly demonstrated, the diverse facets of communication and collaboration in an academic curriculum can be daunting.

In order to prepare teachers and students better for agile-based courses, researchers studied the psychological aspects of teaching agile and Scrum to students. The unfamiliar technical components of Scrum: sprints, Scrum master, stakeholders, etc, were gradually apprehended by most students during the first sprint. Feedback from students and analysis of agile-based courses presented apparent challenges in communication and collaboration, two core agile principals.

A proposed example solution was to take a step back from the traditional software-based outlook of agile, and hone in on agile's human-factors. Students were required to train their social skills during the course. The course introduces an assignment, an interactive game developed specifically for interpersonal skills, using agile. Using such techniques, even in the demanding field of software development, can establish promising results [3].

Since its introduction in the software development community, agile has established a firm reputation as being easily adoptable. Its extensive implementation throughout the software industry has demonstrated its robustness. Like any approach, agile has its many hindrances. Recent research on the various challenges facing organisations looking to implement agile methodologies, show a drift in the focus of the difficulties, from technical practices to limitations, scaling and organisational structure [4].

In the dawn of agile, delegating responsibility among team members seemed daunting and misunderstood. Today, allocation of tasks for an agile team is nothing more than trivial. Many tools have been developed over the recent years, aiding developers in inter-team communication, task assignment, sprint planning, and even between the various stakeholders and Scrum masters. These tools not only promote workflow, they also make the transition for someone new to agile less complicated to understand and undertake [5].

As agile matures, its drawbacks become less certain and more difficult to identify. Based on a study conducted during 2013 and 2014, the researchers concluded that further inquiry would be necessary to uncover the core reason behind the industry's sentiment towards agile's possible setbacks.

Though it took quite a while before agile development methods were noticed by academia, there are now many courses offered at universities all over the world that teach the agile approach. Schroeder et al report on the setup of two software development laboratories specifically designed, so that the social interaction - *how to deal with collaborative software development processes and their need for self-organization, motivation, and work coordination* - is taken into conscious consideration [6].

Scharf and Koch present the design of a Scrum based undergraduate software engineering course, which is based on role-playing. To ensure that their students understand *agile* thoroughly, they designed the complete course as realistically as possible [7]. For instance, they make research assistants play customers and a student plays the Scrum master. Stettina et al discuss their experiences with teaching software engineering practices and their continuous improvement. They emphasise the importance of an intensive coaching routine based on agile practices [8].

Anslow and Maurer report about their experiences teaching a group-based agile software development project course [9]. They report that students enjoy the interactive exercises and the self-organised teams, but also that it is challenging getting the right customer involvement. Rico and Sayani present their results teaching student teams agile methods in their final year course. They report that those teams finding the optimal balance of customer collaboration, technical programming and use of agile methods had better productivity [10].

Kropp, Meier and Biddle report that collaboration between team members, customers, users and stakeholders is a very important part of agile software development [11]. Maturity leads to collaboration; i.e. mature agile teams apply more collaboration practices. Figure 1 shows the influence of agile dimensions.

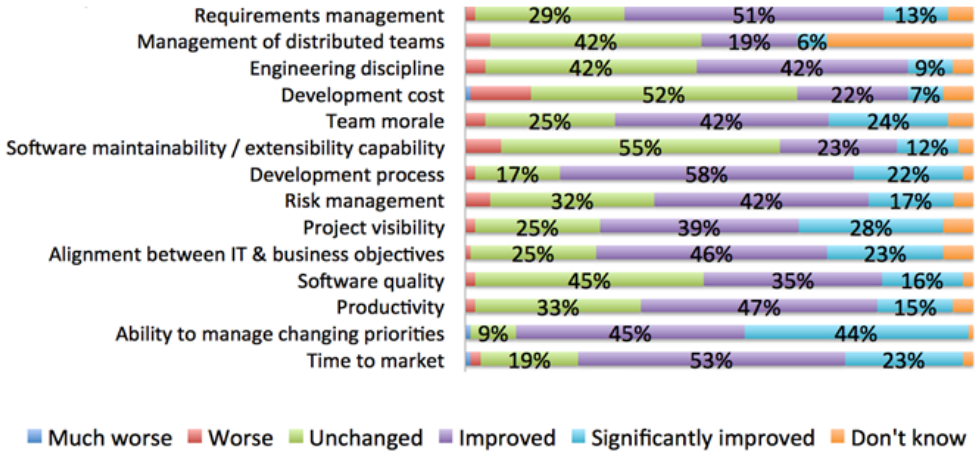

Figure 1: Influence of agile methods [2].

SOFTWARE PROJECT MANAGEMENT COURSE

A course was directed towards undergraduate software engineering students during the second semester of their sophomore year (of a four-year curriculum). The underlying goal of the course was to present indispensable principles of software engineering: 1) the software management process; 2) software validation; and 3) software maintenance [12].

The students familiarised themselves with elementary principles from a course taught during the previous semester that focused on: 1) software configurations management; 2) software construction; 3) software design; and 4) software infrastructure. Sophomore students are a notable choice for such an experiment: students have ample familiarity with the academic and technical arenas, yet, still lack much experience. In other words, sophomore students have a solid foundation with no realised structure. The course differs from a typical academic course, in which material is organised and delivered to students directly from a lecturer on a regular basis. Lectures and course work are precisely scheduled to maximise the potential for a student to absorb and properly interpret the relevant material. However, what the authors provided was a project-oriented course, where the main focus is on the student's ability to self-teach [13].

COURSE OBJECTIVES AND LEARNING OUTCOMES

The course was assembled as an effort to probe the benefits and potential drawbacks of the intended learning outcomes methodologies [14]. A detailed list of objectives and learning outcomes, was introduced before commencement of the semester. Each defined goal targets a different aspect of the course's overall curriculum. The goals can be characterised under two main branches: technical application of learning and theoretical absorption of the abstract material.

This document attempts to avoid the technical details specifically pertinent to the software industry and focuses its attention on the underlying techniques and procedure used to actualise the learning outcomes. The following paragraph will briefly audit each learning outcome; a comprehensive analysis of the methods chosen to materialise the learning outcome, as well as conclusions and results can be found in a later section of this document:

a. Teamwork, collaboration and communication: the drafting of groups was random and indiscriminate of diligence and studiousness. Scrum seemed to be a perfect synergy between project management and soft-skills. Moreover, students were not necessarily grouped with their acquaintances or friends; thus, incorporating the need to cope with difference of opinions and approaches.
b. Self-learning: in this course, the focus is on the self-learning of the students in many aspects: server side, client side, development, programming languages, tooling, task management, and more.
c. Responsibility: it is common, especially among young adults, to dodge criticism and transfer responsibility or fault to others. Students are expected to convey their challenges and potential obstacles with the lecturer or mentor and accept that mistakes are inevitable. By cloaking mistakes, students are ignoring their vast educational benefits.
d. Object-oriented programming (OOP): all students were obligated to take and succeed in a prerequisite course in OOP in semesters prior. Nonetheless, hardly any students had formal, hands-on experience in OOP. Students were also never presented a project on such a scale; both in terms of the amount of necessary coding, as well as the comprehensive understanding essential in scaffolding an OOP project.
e. Problem recognition and solving: with the advent of Internet search engines, problem-solving has become a mandate click of a button or the coined, *Google it!* Unfortunately, the Internet has slowly become bloated with irrelevant or misleading information. Students are meant to acquire a keen insight to properly filter current information from the unreliable majority. Furthermore, they are expected to acknowledge their own problems without an outsider pointing them out. Diligent research in today's digital world is a crucial trait that every student must obtain.
f. Implementation and design approaches: groups were granted the freedom to develop their projects using any programming language. They were also extended the latitude to model their own design approach: *who?*, *what?*, *why?*, *when?* and *how?* will the project be implemented.
g. Adoption of contemporary workflows and tooling: prominent and established tools in the software industry were utilised by the students to promote productivity. Students were not given explicit instructions on how to use the tools, they were recommended on-line tutorials or guides (in English,) to walk them through the basic use cases.
h. Improve reading and writing in English: as non-native English speakers, the majority of the students have seldom been exposed to the spoken English language, and all the more so, written academic compositions. There are students who immigrated from France and Russia, further challenging their comprehension of English as a third language.

METHODS

Course Details

The course's project was to be developed using agile methodologies, (which are beyond the purview of this article) using tools and workflow techniques found in the technology and software industry today [6]. It spanned 13 weeks and was split into four sprints.

a. Lifecycle: for the course was based on the principles of Scrum.
b. Tools: students were granted the option to choose between various tools and project management software. The authors suggested the use of Slack [15] for all correspondence between team members.
c. Source control: students were required to utilise source control for all the project resources: they were allowed to make a choice between Github [16] or Microsoft's team foundation server (TFS) [17].
d. Project management: the project's sprints were managed using waffle.io [18] or TFS.

A teacher's assistant (TA) acting as a mentor was appointed to oversee and usher the progress of the student's projects. Simultaneously, students received the project requirement from the lecturer, who played the role of product owner. By examining the approach and outcomes of the student's projects, the authors were able to better understand the mechanisms students use to learn and how students fail or flounder. A questionnaire was distributed to the students towards the end of the semester to solidify or counter the original hypothesis. Several high-level, abstract lectures were arranged during the semester by the lecturer, as an effort to point students in the right direction. Throughout the whole development process, both the lecturer and TA were privy to all communication between team members and their project management documentations.

A group of 50 students were randomly divided into ten groups of 5-6 students, one of which carried out the role of a group leader. Group leaders were chosen based on their overall grade point average (GPA) from previous semesters, whereas the rest of the students were arbitrarily selected. Each group was assigned a software development project with exactly the same requirements. Groups were entitled to adopt their own development milestone, means of implementation, as well as overall planning and structure of functionality. A weekly meeting between each group (as a whole) and the mentor took place to review past productivity and establish a path for the upcoming week, similar

to Scrum's stand up meetings. The mentor utilised the meetings to survey previous choices that may have led groups astray and struggles they may currently face.

RESEARCH QUESTIONS

What were the objectives of the course? 1) soft skills: communication, teamwork, problem-solving, presentation skills, abstract thinking, self-learning; 2) professional knowledge and tooling: resource management and source control, object-oriented programming, design, application of prior knowledge, contemporary tooling, practical application of theory; and 3) practical experience in a simulated real-world environment.

How were the objectives achieved? 1) using project-oriented learning technique; 2) combination between formal and self-learning; 3) management tool; and 4) advanced practical toolbox (TFS, Github, waffle.io, Slack).

Can one measure the course contribution? 1) exercises and examinations; 2) survey (feedback from students); and 3) internal feedback, self-perception vs. actual status (future improvements).

RESULTS

A Likert scale is an ordered scale from which respondents choose one option that best aligns with their view. It is often used to measure respondents' attitudes by asking the extent to which they agree or disagree with a particular question or statement. Cronbach's alpha *(α)* is a measure used to assess the reliability or internal consistency, of a set of scale or test items. In other words, the reliability of any given measurement refers to the extent to which it is a consistent measure of a concept, and Cronbach's alpha is one way of measuring the strength of that consistency.

Cronbach's alpha is computed by correlating the score for each scale item with the total score for each observation (usually individual survey respondents or test takers), and then, comparing that to the variance for all individual item scores, where: *k* refers to the number of scale items, refers to the variance associated with item *i*, and refers to the variance associated with the observed total scores.

Although the standards for what makes *α, good α* coefficient are entirely arbitrary and depend on one's theoretical knowledge of the scale in question, many methodologists recommend a minimum *α* coefficient between 0.65 and 0.8 (or higher in many cases). Alpha coefficients that are less than 0.5 are usually unacceptable. In this case:

$k = 22$, $\sigma_{y_i}^2 = 28.50$, $\sigma_x^2 = 117.192$, $\alpha = \left(\frac{22}{22-1}\right)\left(1 - \left(\frac{28.50055}{117.1919}\right)\right) = 0.79$

The following section presents the results of the above mentioned survey as they pertain to our overall course goals.

Teamwork, Collaboration and Communication

Were the group's responsibilities divided equally among the team members? Only two thirds of the students thought that responsibilities were divided equally (63%, 4 or 5 on a Likert scale of 1-5).

Did each group member perform their allotted tasks? Around two-thirds of the students felt that each group member performed the task assigned to them (67%, 4 or 5 on a Likert scale of 1-5).

Was there a pleasant atmosphere among group members? More than 72% of the students felt that the project progressed under a pleasant group atmosphere.

To what extent would you like to have an impact in choosing fellow team members? 77% of the students would want to have more influence on team selection.

What degree of difficulty did you face when integrating various coded components? At the point of merging codebases, most of the students did not encounter any difficulty (only 37% of the students felt difficulty).

To what extent did the course contribute to your ability to work in a team? Less than 60% of the students felt that this course improved their ability to work in a team.

Project Management and Self-learning

How much do you value the usefulness of the course in your future professional career? Only 63% of the students believed that the course would benefit their professional future.

To what extent did the course improve your programming abilities? Less than 60% of the students felt that this course improved their programming ability.

To what extent did you succeed in managing course objectives, both the development and planning? About 37% of the students thought that they were able to meet the objectives/schedule of the course.

To what extent does the practical aspect (coding) contribute to the course? More than 72% of the students believed that the applied part contributes to their success in the course.

Tooling

How difficult was it to learn the use of Github/TFS? Over 77% of the students agreed that the use of management tools are not difficult.

In what degree did you utilise the various project management tools? About 42% of the students agreed that they utilise the various *project management* tools effectively.

Did the use of the tool help you better plan the sprint? Less than 40% of the students agreed that the management tools helped them to plan the sprint better.

To what extent was the tool efficient in regard to team work? About 55% of the students thought that the tool was efficient in regard to team work.

To what extent was the tool efficient in regard to communication among team members? Only 38% of the students thought that the tool was efficient in regard to communication among team members.

To what extent was the tool efficient in regard to software development process? About half (~50%) of the students believed that the tool was efficient in regard to software development process.

CONCLUSIONS

Some of the results were contradictory and further research is needed to improve understanding of their justifications. Based on the results, the majority of the students sought a larger impact in choosing fellow team members (77%); however, they felt a pleasant atmosphere among group members (72%), believed that responsibilities were divided equally among the team members (63%) and each group member performed their allotted tasks (67%). One of the main aspects of the agile methodology is communication and flexibility among team members. The fact that students desired a larger role in choosing their groups even after vouching for a pleasant and fairly distrusted work atmosphere is not currently understood.

The TA's subjective opinion, as well as the actually results, actively imply that students enjoy practicality and the actual implementation of their project (73%). Students also believe that the course contributed to their future professional career (63%).

Tooling and source control received an unanticipated review from the students. They were apathetic in regard to the source control tools (Github/TFS) and did not consider them particularly efficient tools for project management (~50%). A suggestion as to why students felt this way can be accredited to their lack of training. A novice programmer, who seldom has the necessity to work on large team-based projects cannot completely fathom the need for source-control or other management tools.

Project-oriented courses offer a mere simulation to the real-world work environment. There are so many volatile and dynamic parameters in the workplace that cannot be induced in an academic course. Tasks related to a given product undergo continuous evolution: market value, consumers' demands, competition and budgets, to name a few, are dynamic in the real world and cannot be conveyed in the rigid frame of a course. Such courses do however grant the student with the essential quality of interpersonal communication, teamwork, and responsibly.

Students also differ from employees when it comes to their ambitions. Students are focused on their grades, the *how* and inattentive to the *why* and *what*. This outlook denies them the ability to absorb the material at hand and be able to apply it further in the future. Employees on the other hand, are focused on their task and are assumed to already have the tools to accomplish them.

REFERENCES

1. Version One. State of Agile Development Survey Results. 01 October 2017, http://www.agile247.pl/wp-content/uploads/2017/04/versionone-11th-annual-state-of-agile-report.pdf
2. Kropp, M. and Meier, A., Swiss Agile Study (2014), ISSN: 2296-2476, FHNW and ZHAW, Brugg, Zurich, 01 October 2017, www.swissagilestudy.ch
3. Kropp, M., Meier, A., Mateescu, M. and Zahn, C., Teaching and learning agile collaboration. *Proc. 2014 IEEE 27th Conf. on Software Engng. Educ. and Training*, Klagenfurt, 139-148 (2014).

4. Gregory, P., Barroca, L., Taylor, K., Salah, D. and Sharp, H., *Agile Challenges in Practice: A Thematic Analysis*. In: Lassenius C., Dingsøyr T., Paasivaara M. (Eds), Agile Processes in Software Engineering and Extreme Programming. XP 2015. Lecture Notes in Business Information Processing, 212, Springer, Cham (2015).
5. Gregory, P., Plonka, L., Sharp, H. and Taylor, K., Bridging the gap between research and practice: the agile research network. European Conference on Research Methods, London, UK (2014).
6. Schroeder, A., Klarl, A., Mayer, P. and Kroi, C., Teaching agile software development through lab courses. *Proc. Global Engng. Educ. Conf.*, 1-10 (2012).
7. Scharf, A. and Koch, A., Scrum in a software engineering course: an in-depth praxis report. *Proc. 2013 IEEE 26th Inter. Conf. on Software Engng. Educ. and Training*, 159-168 (2013).
8. Stettina, C.J., Zhou, Z., Bäck, T. and Katzy, B., Academic education of software engineering practices: towards planning and improving capstone courses based upon intensive coaching and team routines. *Proc. 2013 IEEE 26th Inter. Conf. on Software Engng. Educ. and Training*, 169-178 (2013).
9. Anslow, C. and Maurer, F., An experience report at teaching a group based agile software development project course. *Proc. 46th ACM Technical Symp. on Computer Science Educ*., Kansas City, USA, 500-505 (2015).
10. Rico, D. and Sayani, H., Use of agile methods in software engineering education. *Proc. Inter. Conf. on Agile*, 172-179 (2009).
11. Kropp, M., Meier, A. and Biddle, R., Teaching agile collaboration skills in the classroom. *Proc. 2016 IEEE 29th Inter. Conf. on Software Engng. Educ. and Training*, 118-127 (2016).
12. Bourque, P., Dupuis, R., Abran, A., Moore, J.W. and Tripp, L., The guide to the software engineering body of knowledge. *IEEE Software*, 16, **6**, 35-44, Nov/Dec (1999).
13. Dutson, A.J., Todd, R.H., Magleby, S.P. and Sorensen, C.D., A review of literature on teaching engineering design through project-oriented capstone courses. *J. of Engng. Educ.*, 86, **1**, 17-28 (1997).
14. Adam, S., Using Learning Outcomes. Report for the Bologna Conference on Learning Outcomes held in Edinburgh on 1-2 July (2004).
15. Slack. Where Work Happens, 8 October 2017, https://slack.com/
16. Build for Developers. GitHub Inc. (2017), 8 October 2017, https://github.com/
17. Microsoft. Visual Studio. Team Foundation Server - Share Code. Track Work. Ship Software (2017), 8 October 2017), https://www.visualstudio.com/tfs/
18. Waffle.io. Smart and Simple Project Management (2017), 8 October 2017, https://waffle.io/