Continuous Software Engineering and Unit Testing: From Theory to Practice

HADAS CHASSIDIM
Department of Software Engineering
SCE - Shamoon College of Engineering
Bialik/Basel Streets, Be'er Sheva 84100
ISRAEL

DANI ALMOG
Department of Software Engineering
SCE - Shamoon College of Engineering
Bialik/Basel Streets, Be'er Sheva 84100
ISRAEL

SHLOMO MARK
Department of Software Engineering
SCE - Shamoon College of Engineering
84 Jabotinsky St. Ashdod, 77245 Israel
ISRAEL

Abstract: - With the Agile development approach, the software industry has moved to a more flexible and continuous Software Development Life Cycle (SDLC), which integrates the stages of development, delivery and deployment. This trend has exposed a tendency of increasing reliance on both unit testing and test automation for the fundamental quality-activities during the code development. To implement Continuous Software Engineering (CSE), it is vital to assure that unit-testing activities are an integral and well-defined part of a continuous process.

This paper focuses on the initial role of actual testing – viewing unit testing as a quality indicator during the development life cycle. We review the definition of unit-testing from the CSE world, and describe a qualitative study in which we examined implementation of unit testing in three software companies that recently migrated to CSE methodology. The results from the qualitative study corroborate our argument that under the continues approach, quality-based development practices such as unit testing are of increasing importance, lacking common set of measurements and KPI's. A possible explanation to this may be the role of continuous practices as well as unit testing in the software engineering curriculum.

Key-Words: - Unit testing, continuous practices, continuous testing, integration testing, CICD

Received: May 17, 2021. Revised: August 4, 2021. Accepted: August 16, 2021. Published: August 30, 2021

1 Introduction

From the Agile development approach to the continuous approach, in order to achieve quality, companies evolve their software development practices over time. Typically, most companies follow a particular pattern as their evolution path, often referring to it as "the stairway to heaven" [1]. The pattern implies a dependency between development stages and practices by presenting each as a foundation for the next. Continuous Software Engineering (CSE), also known as 'continuous practice', has become widespread in many software development organizations [2]. This trend enables developers to provide continuous and earlier delivery of adaptations and changes to the software product [2] [3]. The transition to continuous methods has been extensively researched in recent years and

requires the expansion of research related to CD / CI formats and anti-patterns [4] [5]. Continuous Integration (CI) is a practice in which members of a team collaborate and integrate their work frequently. CI development environments enable frequent integration of new or revised code into the mainline codebase [6], leading to multiple integrations per day [7]. Each integration is verified by an automated build, which includes testing to detect integration errors as quickly as possible. Hence, many teams find that the CI approach leads to significant reduction of integration problems and allows more rapid development of cohesive software [7]. Furthermore, the CI approach can reduce the amount of rework that is needed in later phases of development and can speed up overall development time via automated processes. Although it has received very limited attention from the research community, continuous integration, which includes compilation, building, and testing of software, is emerging as one of the success stories in automated software engineering [8] [9]. Continuous Deployment (CD) takes the continuous approach one step further by automatically deploying software changes to production [1]. The CD approach emphasizes build-and-test automation [10] together with a much-reduced scope for each release. Our interviews with 15 information and communications technology companies revealed the benefits and obstacles to continuous deployment [1]. Despite understanding the benefits, none of the companies had adopted a fully automatic deployment pipeline [11]. The study also reveals that adopting continuous deployment practices involves coordination from teams throughout the organization and the domain in which a company operates. The continuous approach stresses the need to explore the impact of quality and testing procedures on all dimensions of the organization and the development process. These begin with the role and responsibilities of "unit testing" in the CSE world [12] [13], continue through effective regression testing techniques [6], and end with test automation, which is considered to be a critical prerequisite for continuous development [14] [10].

Software quality aspects, metrics and measurements have become essential in the context of modern software development processes. ISO 8042 [15] defines quality as "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs". According to the Forrester report [16], companies have shown interest in connecting an "… organization's business to its software delivery capability" by gaining "…a view into planning, health indicators, and analytics, helping them collaborate more effectively to reduce waste and focus on work that delivers value to the customer and the business". This may be considered too broad a definition, as it relates to quality in general. In software development, quality should be more specifically defined by constructing measurements and metrics that indicate or contain the properties to be considered. The widely adopted ISO/IEC 25010 standard [17] determines which aspects are to be taken into account when evaluating the quality of a software product. In fact, an attempt to investigate the Agile method's critical quality factors and measurements [18] fails to identify a single common international standard adopted by the industry. Therefore, addressing the full scale of software quality issues in ASD requires a holistic approach for providing connections among all software development activities, including aspects such as business and development (BizDev) and development and operations (DevOps) as well as integration.

ISO/IEC/IEEE 29119 addresses software testing for quality in five realms: Concepts and Definitions, Test Processes, Test Documentation, Test Techniques, and Keyword-Driven Testing [19]. In the Agile approach, which demands tight connections among all software development activities to be done within a 'timebox' cycle, it is difficult to differentiate between the testing activities within the timebox development cycle. Furthermore, testing activities in Agile are done internally as part of the development routine and by the same people, who are not necessarily testers; which might be the reason for the unclear division between testing levels [20]. Prechelt et al., [21] in a case study, asked the question who does perform the testing and who evaluates outcomes in Agile environments. They found that developers manage to fulfill the responsibilities of conventional testers role by identifying which aspects can be covered by automated testing; meanwhile evaluations are done implicitly by end-users. Hence, an important aspect of testing within the Agile cycle is efficient implementation of test automation [14] [10]. Indeed in the 'DevOps' software development life cycle approach (a current form of Agile) the continuous nature of the cycle is stressed: once development is tied into the release process, monitoring should come in the form of feedback which initiates new planning actions for continuous development [2] [22].

An attempt to model benchmark implementations for continuous integration (CI) [1] concluded that there was currently no consensus on CI as a single, homogeneous practice. Simply stating that a study uses CI is insufficient since it fails to define what

kind of CI is used. Considering the dramatic differences in effects experienced [1], it is necessary to determine the advantages and disadvantages of the various aspects of CI. To that end, and based on the findings in their study, the authors proposed a descriptive model for better documentation of continuous integration variants. Shahin et al. [2] provided a systematic review of approaches, tools, challenges, and methods identified in empirical studies on continuous practices. Sixty-nine papers were selected from 2004 to 2016 for data extraction, of which 56.5% were published in the last three years, only four of which addressed improvements in software testing. However, 39% of the studies mentioned that testing efforts and time are critical factors. Not even a single study dealt with the effects of transitioning to CI and the corresponding implications for quality and testing. Nilsson et al., [23] focused on the question of visualizing end-to-end testing activities in order to support the transformation towards CI. By end-to-end testing, they refer to all code, from code written by individual engineers to product release. The aim of that research was to gain insights into how to support the transition towards continuous deployment in the software development industry. Their case studies had some challenging findings about change processes: significant duplicated testing efforts, slow feedback loops, late testing of quality attributes, no overview of testing in commercial companies, ad-hoc testing or tactical improvement efforts. These findings indicate a lack of a holistic, end-to-end understanding of testing activities and their periodicity. The product of that research was the creation of the Continuous Integration Visualization Technique (CIViT) and an attempt to implement it for the case study companies. Our work looks at this issue from the deeper perspective of the testing activities themselves and examines the impact made on the basic layer, namely, unit testing and related activities. In software development, unit testing is a crucial link in the chain of quality activities which aim to improve an organization's outcomes by focusing on quality goals and recruitment needs as early as the initial programming stage. However, systematic guidelines are not widespread, and an acceptable definition of the term "unit testing" [12] is essential. Unit testing is a term which describes the action of the programmer when testing an isolated, atomic, and code-related portion of software (a unit) [12]. The motivation of this study was to examine whether the transition to continuous methods changes the perception about the necessity and importance of the testing schemes in general and unit testing in particular. In order to examine the implementation of unit testing in the continuous practices of different software industries, a case study of real-life implementations of unit testing needed to be conducted. The present work is an exploratory study meant to identify the benefits of the continuous approach with respect to unit testing and related activities. Our findings describe actual practices and generate hypotheses more than they confirm them. Section 2 elaborates the concept of unit testing, its role, and its common definition in the development environment. Section 3 reports on a case study of three participating software companies and their real-world applications of unit testing in CSE. Our paper concludes with a generalization of the results of all the cases reported and a presentation of guidelines and recommendations for the software industry.

2 Definitions for 'unit test'

The definition of the key term "unit test" is neither clear nor precise. Apparently, the term 'unit test' can be interpreted and implemented in different ways for different application domains. Furthermore, the term, as reflected in the analysis of academic literature [12], seems to suggest two definitions:

- The classic way (66%). – About two-thirds of the examined academic literature refers to 'unit-tests' as the testing action of the smallest, isolated, atomic, unit of code. Such testing is done mainly by the developer.
- The component way (24%). – The focus is on a unit–of-functionality, whether or not it is seen as the smallest, indivisible portion of the program. Here, unit testing is administered mainly by testers.

Apart from the distinction between "unit" and "component", and who actually performs the tests, and beyond the understanding that once defined it can be detached from the code, a clear definition of the specific usage of this terms is required. In fact, this requires two more main distinctions to be made. First, a distinction between the action which delineates the content of the item to be tested and the process as an execution effort, i.e. the actual application of the test. And second, a distinction of levels of abstraction: between actions relating to execution of testing for a unit of code, and the more general description of testing operations on a portion of the program, which may be expressed in a functional or business terminology.

In light of these needs and the growing importance of the role of the level of unit testing, Chassidim et al., [12] recommend that the following categories of

testing be distinguished in the early stages of software development:

• **Unit test** – refers to the structural (atomic, isolated and code-related portion of the software) aspect of the unit that passes the testing action - i.e. an action which presents the content of the item to be tested. Common practices include incremental writing of unit tests using Test-driven development (TDD) or Test-Driven Maintenance (TDM) methods [24].

• **Unit testing** – refers to the process of testing the isolated, atomic, and code-related portion of the software (a unit). I.e. the process as an execution effort and refers to the use of xUnit testing, usually, the developers perform this activity by themselves.

• **Component testing** – This is the testing of a functional and larger portion of the program (a component). Another set of skills and another kind of knowledge are needed to perform this portion of the work.

In fact, unit testing may form the basis for component testing that can be considered a higher level of testing. Unit testing should test individual behaviors. However, most methods deal with many behaviors. Therefore, a serious pitfall might be encountered when developers test too large a unit or when they consider a method within the software to be a unit. This is particularly true in the case of inversion of control, where unit testing typically turns into end-to-end integration testing.

Component testing is sometimes known as module or program testing. Component testing is done mostly by a test engineer. It may be done in isolation from the rest of the system depending on the model of life cycle development chosen for that application. In such cases, any missing software is replaced by 'stubs' and 'drivers' that simulate the interface between the software components in a simple manner.

A successfully passed test must continue to be administrated as long as the codebase remains constant. Fulfilling ideal code conditions for unit testing includes isolation and atomic code [12] that improves a programmer's understanding of system requirements. A properly written test can be executed on an isolated section of the code and can pass even if the developer did not understand the requirements correctly. As a result, all the tests will pass even when many of them did not actually validate the intended functionality of the code. However, tests that rely on an external application protocol interface (API), network connections, user input, threading, or other external dependencies, must be mocked. Mocking has shown itself to be a proven and effective technique and is a widely adopted practice [12]. For example, if the network connection suddenly becomes disconnected, the code will subsequently fail. A well-established solution is to implement a mock in place of the actual network connection, so that the tests can continue passing. It is vital to separate the two aspects and to allocate the best resources for each assignment or, alternatively, to train the developers to distinguish between a classical definition of unit testing and a mixed one and to provide them with new skills and knowledge, so that they can perform these two categories of testing separately in the early stages of software development [12].

3 Implications on software engineers' education

3.1 Testing topics in software engineering curriculum

Theoretically, from the early days of testing, practitioners distinguish between unit tests and integration tests. However, practically software engineering community rarely distinguishes between these two strategies, mainly because it is not straightforward to separate them in the code repositories under study [25] .The source of the problem may lie in the gap created in the qualification process of the engineer. The Software Engineering (SE) curriculum guideline (SE2014) is widely used by the software engineering educator for the design and modification of undergraduate software engineering programs. An in-depth examination of the common areas in the software engineering education knowledge (SEEK) that are defined by SE2014, refers to the QUA (software quality), only 2%. Furthermore, today less than 10% of the official SEEK addresses quality and testing directly. The relative share of unit testing and integration testing in the certification process is difficult to diagnose [26] [27] [28]. These findings reflect the recommendations of the dedicated task force that reviewed the curriculum guidelines for undergraduate degree programs in software engineering on 2014 [27]. No specific recommendation were raised regarding needed changes in the testing discipline. Another attempt to evaluate the SE curriculum is the SWECOM, which compares between SE2014 and employer's need [29] [30] [31]. SWECOM took a practitioner approach and recommended competency of an entry-level software engineer professional. SWECOM shows that in the Software Testing skill area SE2014 is not matching up to their requirements, especially the skill

set of software testing measurement and defect tracking. The lack of proper attention to the contemporary testing sub-topics might be a result of the long and tedious process that accompanies the modifications of the curriculum in an academic setting [29]. However, when it comes to a dynamic field like software engineering education the long respond for updating might be critical to the graduated student professionalism. The missing testing skills might cause a discrepancy between the skills learned from an SE university education and those needed in SE employment.

3.2 Continuous testing in software engineering curriculum

Continuous software engineering is an emerging area in industry that incorporate testing as an essential ingredient. In order to align with the industry needs, it is important to train software engineers in the world of CICD, including continuous testing. A recent study performed a meta-analysis to provide a consolidated view on how to align SE education with industry needs, to identify the most important skills and also existing knowledge gaps [32]. The majority of the papers reported that testing is one of the most important areas with the greatest knowledge gaps between SE education and industry [32] [33]. In addition, we are not aware of a concrete educational program [34] that refers specifically to the world of CICD as well as the continuous testing which is part of it. The absence of the adaptation of common body of knowledge (e.g., SWEBOK, SEEK, SWECOM) to the continuous approach may imply future lack of knowledge in the field.

4 Research methodology

4.1 Interviews

In order to explore the degree of the implementation of a unit test within companies that have adopted the continuous software engineering (CSE) approach, and by assuming diversity of actual practices in the companies, we chose to apply a qualitative 'structured interviews' format [35] [36].

The qualitative research we conducted had five stages. Figure 1 presents the process of preparation and implementation of the research.

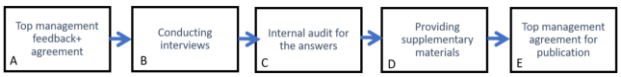

Fig. 1. Preparations for the interview and information collection process

Stage 1 (Fig. 1A). After identification of software companies claiming to implement the continuous approach, an application was made to the executive managers to confirm their company's participation in the study. Once approved, we asked the executive managers in charge of implementing the continuous approach for referrals to relevant role-holders and to ensure high diversity of participants in the interviews. Questions were sent to the managers for internal review and feedback to improve the quality of the interviews. Based on their input we updated the questions.

Stage 2 (Fig. 1B). After consultation and approval from the management of each company, and before conducting the interviews, in order to receive spontaneous answers the interviewee was handed a document with the high-level topics, and just after that we conducted the interviews with the relevant role holders.

Stage 3 (Fig. 1C, D). Before approval for publication, management applied an internal audit of the answers, including a full final transcript of the interviews and supplemental materials to support the portrayal of company processes.

Stage 4 (Fig. 1E). To maintain authenticity, the original terminologies and phrases were digitally recorded and approved before publication.

In all the interviews we adhered to the following principles:

1. Interviews were held with the declared support of the companies' senior management.
2. Interviews were conducted by interviewer with over 30 years of experience in software management at a variety of international software companies.
3. Interviews where held during working hours, and in the meeting rooms on the company's site - a closed and isolated room.
4. The interviews kept the principle of a synchronous conversation with one participant at a time [36].
5. The time allocated for each interview was up to three hours, and for each day of interviews only up to two interviewees were allotted.
6. Each interview was recorded and transcribed and documented.

Further, the interviewer was free to ask 'unstructured questions' - which requires the interviewer to have complete command of the professional language and the ability to direct and lead the conversation back to points that interest us in the research.

The interviewer began by introducing himself and his experience in managing software companies. He stated that the interview was being conducted as part of a broader research project aimed at examining the impact of the continuous process on various aspects of development and especially on the unit-testing phase.
The interviewee was then asked to tell about himself, his professional experience, his role in the company, and his general perception about the continuous process. It was an unstructured conversation that allowed the interviewer to learn and get to know the interviewee better, with the interviewee also allowed to ask questions.
For defining the research questions, we chose to focus on three time phases: Prior to implementation of continuous practices, current situation, and future prediction.
When defining the questions and during the interview we made sure to avoid biases originating from the interviewee, the interviewer or the wording of the questions. For example, we made sure not to present our personal opinions or to indicate how important the unit testing process is for quality. We made sure not to ask questions worded negatively or vaguely, and allowed the interviewee to answer that he does not know.
From the questions above, we wanted to shed light on the organizational culture, roles and perception of importance, regarding unit test prior and after the transition to continues methods. It was interesting to examine whether the transition to continuous methods caused changes in policies, perceptions, scope, methodologies, practices or tools, in the testing process in general and in the unit testing in particular. And if so, we wanted to understand why and how it was expressed. This qualitative research is important for integrating current industry topics into software engineering study programs, especially in courses related to software quality assurance.

The following questions were presented to the participants referring to these aspects:
Q1: Prior to the implementation of continuous practices in your company, what was the company's approach toward unit tests?
Q2.1: What is the current approach towards unit testing?
Q2.2: What tools and technologies are used for unit testing?
Q2.3: What tools are used for unit testing automation?
Q2.4: What are the other testing performed?
Q2.5: What are the measurements and KPIs used in the organization and for unit testing and quality validation?
Q3: How does the organization view unit testing in the full context of the process and product quality?

4.2 Participants

Interviews were conducted at the sites of three leading software development companies in Israel and the USA (n=15). We selected participants on the basis of availability and position in the company to gain different perspectives (as shown in Table 1).

	Company#1	Company#2	Company#3
Domain	Financial crime, risk, and compliance solutions	Network equipment manufacturers service providers	provides infrastructure and maintenance services
Size	Large size, 5K + Employees	Mid-size, 100+	Large size, 250
Cycle time & increments time	3 weeks; 3 months	3 weeks; not fixed time	Not fixed time
Top Management	CQE, Division GP	CIO, CTO	CIO, director of IT
Process Management	CICD PM	CICD PM	CICD PM
Technical Level	2 TL, SW architect	TL, SW architect	Senior line programmer

Table 1 Interviewee distribution

Companies were selected from varying domains and business sizes to provide a rich characterization of their implementation of unit continuous development. Companies had previously reported that they were already working in a continuous environment.

4.3 Data analysis

In this study we adapted the relatively known method of Critical Discourse Analysis (CDA) [37] [38] [39] to analyze the data gathered during the interviews. Discourse analysis is a collective name for a number of scientific methodologies that analyze how meaning is created and communicated though written, vocal or sign language. Discourse analysis is used in many disciplines, mainly in the social sciences, to learn about contemporary processes of social transformation. However in this study, CDA

was applied in a software development context to analyze the discourse gathered during the interviews descriptively approach. This method is more exploratory and can help better understand the changes and impact of CICD on unit testing, relying on the knowledge and experience of interviewees rather than determining in advance the ideal approach. The analysis was adapted from Fairclough's transdisciplinary CDA [39] that referred to two phases. The first phase deals with the selection of the research topics, providing previous discourses. In our case, we considered past findings about unit test definitions [12] and connected them to the practices of unit testing in the CICD world, since we assume that the unit testing practices have changed. The second phase continues with the selection and the analysis of current text (i.e. transferred vocal to text). Interviewees' answers were compiled into sections or groups of information, and categorized in order to collect consistent phrases, expressions, and ideas common to research participants [35] [40]. Data from recorded interviews were then color-coded according to the ideas raised without determining in advance which is preferred. We are aware of the limitation that interpretations are subject to a researcher personal preferences. To minimize bias, this evaluation was applied separately by two researchers of the study and also by a representative of each company.

5 Case study findings

This section summarizes the responses and relates the insights derived from all the cases. The three selected case studies are intentionally different to include a diverse and broad picture. All three companies that participated in this study reported that they had already adopted unit testing as a vital part of their development scheme. Moreover, the unit testing role was augmented and the CICD project positioned unit testing activities as central to all activities related to quality (see appendix 1: A summary of replies from three companies)

To summarize, companies adopt unit testing automation as their main automation engine and define it via the classical definition (see Section 2.2) in which X-unit is their main unit testing tool and environment. Test automation [14] is an important factor when addressing testing during the CICD project. Companies select their testing tools by understanding their need for automation. They would not use an X-unit tool that cannot provide a test automation [14] infrastructure. However, since most of them are aware of the need for component and integration testing, they may look for additional testing infrastructure to fulfill testing automation requirements for the full CICD activities.

Two companies (1 and 2) have maintained additional testing levels internal and external to the Agile development cycle, although there is a general tendency to assure their testing level coverage within the Agile development cycle. This by itself illustrates the preservation of expertise within the development team.

All the companies identified measurements and quality goals as their weak spots. They all felt the lack of well-established criteria for evaluating and planning their activities. The leading complaint addresses the coverage issue – mostly what percentage of the code should be covered during unit testing.

While unit testing was found to be a prominent quality component, other testing processes such as integration testing, customer orientation and etc., were found to be less significant. A possible explanation could be the ease and availability of making changes and improvements according to the ongoing approach.

Additional issues raised by the interviews

Outcomes reveal relatively mature, well-managed processes among the three companies. Regardless of the different solutions and implementation of CSE projects, we may generalize our impressions as follows. The transition to CI is a large organizational project which requires management support. Automation is a must; without it there is no way to perform continuous development. Unit testing also plays an important part in achieving the desired quality and is becoming an activity done on a daily basis by programmers.

Although the formation of the desired infrastructure could be specified, hardly any single tool provides a full solution. Each company selects and assembles its own tools and integrates all of them. Another aspect became evident in our study: the responsibility for quality assurance is being transferred to development teams. However, we did not find the measurements, tools, and standards to be mature at this point. Moreover, the economic benefits of the different testing levels have yet to be formalized.

The distinction between unit testing and component testing (as appears in Section 2.3) is supported by our findings. Different people are assigned to design and execute unit testing and component testing. The former is developed and executed by developers, while the latter is developed and executed by testers and experts. A possible explanation for this is that component testing is partially manual work and requires different test automation tools [10]. .

6 Discussion and conclusions

Looking at CSE trends through these case studies illustrates the importance of unit testing activities and outcomes for modern software development. The participating companies associated unit-testing activity with the image of quality; thus intensive execution of unit testing was perceived as a prominent quality indicator. All of them regarded it as an essential ingredient for assuring the quality of their product. Therefore, the broad adaptation of unit testing in the CSE may improve the quality and reduce the development cost control. However, to significantly contribute to quality management, the testing processes throughout the pipeline require adequate measurements and standards. Most of the interviewees reported that measurements and specific targets had yet to be implemented.

Further, while unit testing by itself is a key to achieve quality improvements it does not address all the complexities and the contributions of other testing levels such as integration testing and non-functional testing. The results of this study show that unit testing "is done by the developers with the assistance of testing experts within the agile development teams". On the one hand, having testers work together with programmers improves the communication, shortens the time to transfer information and can help with locating the faulty code, since the team itself is involved in the development of the code. On the other hand, results indicate a possible blurring of boundaries between the different levels and types of tests. A possible explanation for this finding is that CICD as a prominent CSE practice, forces a single unit to perform all the related activities while developers are familiar with the unit testing rather than other testing levels. Training of team members to have both unit testing proficiency and be able to clearly distinguish between different testing activities is thus essential.

Team members in modern software development projects are required to perform new activities, which were done previously by specialists. For example, a key success factor is the ability to automate all activities previously done manually, including many types of testing: regression, performance, security, privacy, adaptability, deployability. Another aspect, although not previously discussed, is "exploratory testing", a means of improving quality that is not considered in the new development cycles. Programmers working in isolation often do not have the expertise to practice this form of exploratory testing or to use the technique to maximum effect. That said, an experienced tester working with a programmer in an Agile team can make very effective use of this technique to the benefit of the quality of the code produced. Based on our findings, we are concerned that teams might not have the skills or be ready to comprehend the new and complicated needs which go beyond unit testing.

We suggest a further examination of the findings by increasing the number and the diversity of the participated companies around the globe. Additionally, a deep understanding of the long term planning, training, and organizational implications of these changes is an essential element for CSE migration. It is possible that some testing and quality expertise might decrease when professionally-oriented team-members are re-allocated within the new structure of an organization. Organizations should strive to preserve the continuity of knowledge and expertise during the move to CSE.

Based on the common knowledge foundation for software engineering education and practices, we found that there is a significance difference in the way SWEBOK and SEEK treat unit testing and integration testing. While SWEBOK distinguish between unit and integration testing and refers to both of them, there is a clear bias toward unit testing. Unit testing is mentioned in SWEBOK twice as much as integration testing. However, SEEK assigns only a slight part of the topics for quality, and does not differentiate between unit and integration testing. Our findings point to the need to evaluate and update the SE education program and to strengthen testing knowledge and skills. Particularly, we recommend to update the academic curriculum with integration testing topics. We, (as educators) imprint our newly born software engineers so they may influence CICD projects. The outcome of this research may lead to transition to a new SE teaching curriculum, empathizing the additional testing aspects supporting CICD.

References:

[1] H. H. Olsson, H. Alahyari and J. Bosch, "*Climbing the" Stairway to Heaven"--A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software*." in 38th euromicro conference on software engineering and advanced applications, 2012.

[2] M. Shahin, L. Zhu and M. A. Babar, *"Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and*

practices." IEEE Access, vol. 5, pp. 3909-3943, 2017.

[3] D. Ståhl and J. Bosch, *"Modeling continuous integration practice differences in industry software development"* Journal of Systems and Software, vol. 87, pp. 48-59, 2014.

[4] M. Hilton, N. Nelson, T. Tunnell and D. Marinov, *"Trade-offs in continuous integration: assurance, security, and flexibility."* Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 197-207, August 2017.

[5] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall and M. Di Penta, *"An empirical characterization of bad practices in continuous integration."* Empirical Software Engineering, vol. 25, no. 2, pp. 1095-1135, 2020.

[6] S. Elbaum, G. Rothermel and J. Penix, *"Techniques for improving regression testing in continuous integration development environments"* in the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014.

[7] M. Fowler and M. Foemmel, *"Continuous integration. Thought-Works."* 16 May 2018. [Online]. Available: http://www. Martinfowler.com/.

[8] M. Hilton, *"Understanding and improving continuous integration."* in 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016.

[9] M. Hilton, N. Nelson, T. Tunnell and D. Marinov, *"Trade-offs in continuous integration: assurance, security, and flexibility."* in 11th Joint Meeting on Foundations of Software Engineering, 2017.

[10] J. Kasurinen, O. Taipale and K. Smolander, *"Software test automation in practice: empirical observations"* Advances in Software Engineering, 2010.

[11] M. Leppänen, S. Mäkinen, M. Pagels, V. Eloranta, J. Itkonen, M. V. Mäntylä and T. Männistö, *"The highways and country roads to continuous deployment."* IEEE software, vol. 32, no. 2, pp. 64-72, 2015.

[12] H. Chasidim, D. Almog, D. B. Sohacheski, M. L. Gillenson, R. S. Poston and S. Mark, *"The Unit Test: Facing CICD-Are They Elusive Definitions?"* J. Inf. Technol. Manag., vol. 29, no. 2, pp. 40-54, 2018.

[13] K. Naik and P. Tripathy, Software testing and quality assurance: theory and practice. John Wiley & Sons. 2011.

[14] K. Wiklund, S. Eldh, D. Sundmark and K. Lundqvist, *"Impediments for software test automation: A systematic literature review"* Software Testing, Verification and Reliability, vol. 27, no. 8, 2017.

[15] ISO, *"ISO 8402 for quality management and quality assurance–Vocabulary."* Standard, ISO. 1994.

[16] S. Martínez-Fernández, A. M. Vollmer, A. Jedlitschka and et. al., *"Continuously assessing and improving software quality with software analytics tools: a case study"* IEEE access, vol. 7, pp. 68219-68239, 2019.

[17] ISO/IEC, *"ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.,"* Standard ISO, 2011.

[18] G. Arcos-Medina and D. Mauricio, *"Aspects of software quality applied to the process of agile software development: a systematic literature review."* International Journal of System Assurance Engineering and Management, vol. 10, no. 5, pp. 867-897, 2019.

[19] H. Alaqail and S. Ahmed, *"Overview of software testing standard ISO/IEC/IEEE 29119."* International Journal of Computer Science and Network Security (IJCSNS), vol. 18, no. 2, pp. 112-116, 2018.

[20] R. Black, *"Certified Tester Foundation Level Extension Syllabus Agile Tester,"* 2014.

[21] [21] L. Prechelt, H. Schmeisky and F. Zieris, *"Quality experience: a grounded theory of successful agile projects without dedicated testers."* in 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016.

[22] R. Pietrantuono, A. Bertolino, G. De Angelis, B. Miranda and S. Russo, *"Towards continuous software reliability testing in DevOps."* in 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), 2019.

[23] A. Nilsson, J. Bosch and C. Berger, *"Visualizing testing activities to support continuous integration: A multiple case study"* in International Conference on Agile Software Development, 2014.

[24] E. Shihab, Z. M. Jiang, B. Adams, A. E. Hassan and R. Bowerman, *"Prioritizing the creation of unit tests in legacy software systems."* Software: Practice and Experience, vol. 41, no. 10, pp. 1027-1048, 2011.

[25] G. Orellana, G. Laghari, G. Murgia and O. Demeyer, *"On the differences between unit and integration testing in the travistorrent dataset."* In 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017.

[26] M. Ardis, D. Budgen, G. W. Hislop, J. Offutt, M. Sebern and W. Visser, *"SE 2014: Curriculum guidelines for undergraduate degree programs in software engineering."* Computer, vol. 48, no. 11, pp. 106-109, 2015.

[27] D. Budgen, *"Applying the SE2014 Curriculum Model"* in 2015 IEEE 28th Conference on Software Engineering Education and Training, 2015.

[28] H. Chassidim, D. Almog and S. Mark, *"Quality Development (QDev) Unit in Software Engineering School."* World Transactions on Engineering and Technology Education (WTETE), vol. 16, no. 3, pp. 249-253, 2018.

[29] M. Towhidnejad, O. Ochoa and A. Kiselev, *"An Analysis of the Software Engineering Curriculum Using the Guideline Models."* 2020.

[30] R.E. Fairley, *A Software Engineering Competency Model (SWECOM).* IEEE Computer Society, 2014.

[31] M. Towhidnejad and M. Al Balooshi, *"Determining Degree of Alignment of Undergraduate Software Engineering Program with SWECOM."* in Proceedings of the International Conference on Software Engineering Research and Practice (SERP), 2017.

[32] V. Garousi, G. Giray, E. Tüzün , C. Catal and M. Felderer, *"Aligning software engineering education with industrial needs: a meta-analysis. "* Journal of Systems and Software, vol. 156, pp. 65-83, 2019.

[33] V. Garousi, G. Giray, E. Tuzun, C. Catal and M. Felderer, *"Closing the gap between software engineering education and industrial needs."* IEEE Software, vol. 37, no. 2, pp. 68-77.

[34] P. Bourque and R. E. Fairley, *SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge,* IEEE Press, 2014.

[35] D. W. Turner III, *"Qualitative interview design: A practical guide for novice investigators."* The qualitative report, vol. 15, no. 3, p. 754, 2010.

[36] P. Ralph, S. Baltes, D. Bianculli, Y. Dittrich and et. al., *"empirical standards."* ACM SIGSOFT, 2020.

[37] R. Wodak, "*Critical discourse analysis at the end of the 20th century."* Research on Language & Social Interaction, vol. 32, no. 1-2, pp. 185-193, 1999.

[38] D. Schiffrin, D. Tannen and H. E. Hamilton, *"Introduction to the first edition The handbook of discourse analysis,"* pp. 1-7, 2015.

[39] N. Fairclough, *"Critical discourse analysis: The critical study of language."* Routledge., 2013.

[40] S. Kvale and S. Brinkmann, *"Interviews: Learning the craft of qualitative research interviewing."* SAGE, 2009.

[41] G. Orellana, G. Laghari, A. Murgia and S. Demeyer, "On the differences between unit and integration testing in the travistorrent dataset," 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR) , pp. 451-454, 2017.

Appendix 1: A summary of replies from three companies

Q1: Prior to CI CD project, what was the companies approach to unit testing?

Company 1:

- Unit testing was implemented differently on each team
- Each product line started the project from a different starting point

Company 2:

- Unit testing culture and tradition was already present in all software development teams; actual implementation was left to individual developers and team leaders.
- Having unit testing of Legacy code vital

Company 3:

- The company had adapted unit testing.
- Unit testing was part of a transition to more rapid development
- There was variance between the different lines of business.

- Large legacy software was not tested in unit testing.

Summary:
Overall, unit testing has been implemented prior to the moving to CICD. However, the implementation is varied.

Q2.1: What is the current approach towards unit testing?

Company 1:
There is no clear approach towards the definition of the unit test – most testing should be done by the developers, but since adopting it all testing is part of the agile cycle. They include integration testing as part of the testing and are not clearly differentiating between the various testing activities.

Company 2:
Unit tests remain as major testing instrument. It is done by the developers with the assistance of testing experts within the agile development teams.

Company 3:
Unit test is a mandatory part of a developer's daily work. But unit test infrastructure is not yet internalized and implemented. The unit test adaptation level is determined locally by the development leaders. Unit test coverage is less than 40%.

Summary:
Unit testing is a major component in the SDLC. In most cases, the developers are responsible for it. In one case, it was reported to include integration testing.

Q2.2: What tools are used for unit testing?

Company 1:
All unit testing is done using X-unit testing tools, the company is using different development languages and environment therefore the implementation of repeatable (automated) unit tests is not uniform.

Company 2:
X-unit tools are the major part of the testing. Since a large portion of the software is both cloud oriented or connects directly to hardware, Mocks and other STABs are being extensively used.

Company 3:
Only recently is the company attempting to enforce a specific tools at the different development teams. The common suggested tools are the unit test infrastructure already inherited within the software language – X-unit.

Summary:
X- unit is the common testing environment. In one case, it was mentioned that mocks + stabs were used, which indicates a higher level of testing that includes integration

Q2.3: What tools are they using for unit test automation?

Company 1:
Each environment (Java, C, C#) creates its own test automation infrastructure. So one of the CI CD project team goals is to integrate it all in the implementation train.

Company 2:
Since the developed unit test will be used later on as part of regression testing it is important to have a stable test automation infrastructure. The project has developed a maturity model for automated test stability (unstable/stable test flow).

Company 3:
Test automation mainly consists of repeated executions of unit tests, without strategic planning.

Summary:
Automation includes unit tests without strategic planning, and in one case it is considered as regression testing.

Q2.4: What other testing levels are they following?

Company 1:
Apart from unit testing, the Agile team is in charge of other testing activities – integration and system testing. The company maintains external testing activities such as performance and security which are mostly done as part of later stages on the Agile deployment train.

Company 2:
Unit tests and integration tests are done mostly by quality experts within the Agile team. Other testing levels are being executed – performance, special customer oriented testing etc.

Company 3:
Company #3 does not (officially) have other testing levels. "…We may add another ad-hoc test level which is done when a problem or a challenge is being suspected…" A direct question in regard to performance testing was answered by: it is always a decision between the practical immediate need and quality – and most of the time the practical need wins.

Summary:
Two of the three companies reported about integration testing, which is performed later in the train, usually by an external team. In one case the decision about other testing levels is done on ad-hoc basis.

Q2.5: What measurements and target (if at all) have they for unit test and quality validation?

Company 1:

Measurement and standards are not yet determined by the company. One of project teams has the current task of formalizing measurements, standards and all needed reporting tools for all levels.

Company 2:

Measurements and standards are being debated during the implementation of CI /CD project. Due to the diversity of development languages and infrastructure, the company needs to decide what measurement and reporting will they use

Company 3:

It is following SAFe maturity level measurements, and has targets, timeframes and goals for achieving levels dictated by management

Summary:

In two cases, measurements are not yet determined. In the third case, the company aims to work according to SAFe principles and measurements. No specific targets were mentioned.

Q3: How does the organization sees unit test action at the full context of quality of the process and product?

Company 1:

- Unit testing is mandatory and the most significant act of quality assurance
- There are other important testing activities like integration and system testing.
- Unit testing makes up the largest portion of their test regression packages.
- Other professional expertise is maintained and operates as a central service units.

Company 2:

- The company treats unit testing as important test level
- A contributor to quality and definitely not the only one.

Company 3:

- Unit test and ATD automation coverage are the sole quality concern.
- Quality issue could be solved by the next development cycle.

Summary:

Unit testing was considered as the major contributor of quality. Additional testing levels are conducted by different professional's expertise other than the developer.

Contribution of individual authors to the creation of a scientific article (ghostwriting policy)

Haddas Chassidim, Dani Almog and Shlomo Mark carried out together the entire research.
Haddas Chassidim carried out the Critical Discourse Analysis (CDA)
The interviews were conducted by Dani Almog.

Creative Commons Attribution License 4.0 (Attribution 4.0 International, CC BY 4.0)