**Continuous Software Engineering and Unit Testing**

**from Theory to Practices**

|  |  |
| --- | --- |
| Chassidim HadasDepartment of Software EngineeringShamoon College of Engineeringhadasch@sce.ac.il | Almog DaniDepartment of Software EngineeringShamoon College of EngineeringAlmog.dani@gmail.com |
| Mark ShlomoDepartment of Software EngineeringShamoon College of Engineeringmarks@sce.ac.il |  |

**Abstract**: The software industry has recently moved to a more flexable and continuous Software Life Cycle Development (SDLC) with the Agile development approach, which integrates the stages of development, delivery, and deployment. This trend has exposed the tendency of increasing reliance on unit testing as well as test automation for the fundamental quality assurance of the code development process. In order to implement CSE, it is vital to assure that unit testing activities are an integral and well-defined part of the process. In this paper, we review the academic definition of unit testing from the CSE world and introduce a case study that examines the implementation of unit testing at three software companies that have recently moved to CSE methodology**. The results corroborate the argument that unit testing is a cornerstone for development and an indicator of software quality.**

**Key words: Unit testing, Continious practices, Continuous testing, integration test , CICD**

# Introduction

Continuous software engineering (CSE), also known as continious practice, has become wide-spread in many software development organizations [1]. This trend enables developers to provide earlier and continious delivery of adaptations and changes to the software product [1, 2]. However, it presents a need to understand the impact on quality and testing procedures with respect to all dimensions of the organization and of the development process. In software development, unit testing is one of the primary and basic activities of the development process and is executed by the programmers themselves; therefore, a common language among team members, who use the term "unit testing" [3] is essential.

To examine the implemetation of unit testing in the continious practices of different software industries, a case study of real-life implementations of unit testing must be conducted. Unit testing is considered a crucial link in the chain of quality activities, which aim to improve an organization’s outcomes by focusing their quality goals and recruitment needs as early as the initial programing stage, and unit testing is the term that describes the action of the programmer. This study aims to identify the benefits of the continious approach toward unit testing and the related activities. Section 2 elaborates and explains the CSE revolution, its role, and common definition in the new development environment from the view of unit testing. Section 3 reports on a case study of three participating software companies and their real-world applications of unit testing in CSE. The paper concludes with a generalization of the results of all the cases reported and a presentation of our guidelines and recommendations for the software industry.

# Background

## Software quality

ISO 8042 [4] defines quality as “the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs.” This may be considered too broad a definition, as it relates to quality in general. In software, quality should be more specifically defined by constructing measurments which indicate or contain properties to be considered, for example the widely adopted ISO/IEC 25010 standard [5], which determines which aspects to be taken into account when evaluating the quality of a software product.

A more recent software quality standard, ISO/IEC/IEEE 29119 [6], addresses software testing aspects according to five categories: Concepts and Definitions, Test Processes, Test Documentation, Test Techniques, and Keyword-Driven Testing [6].

Software quality aspects and measurments become even more essential in the context of modern software development processes, such as Agile Software Development (ASD). An attempt to investegate the agile critical quality factors and measurements [7] fails to identify one common international standard adapted by the industry. Nevertheless, these standards are not directly correlated with continuous integration/continuous delivery (CICD), and the incremental nature of these processes produces continuous sources of data (e.g., customer feedback).

According to the Forrester report [8], 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.” Addressing the full scale of software quality in ASD represents a holistic approach to providing tight connections among all software development activities, including aspects such as business and development (BizDev) and development and operations (DevOps) as well as integration.

From Agile development to continuous deployment, companies evolve their software development practices over time. Typically, most companies follow a particular pattern as their evolution path [9], often referring to this path as the “stairway to heaven.”

Figure 1 The “Stairway to Heaven” evolution model [9]

The pattern suggests dependency between the development stages by presenting each one as a building foundation for the next (Figure 1). Several studies have discussed the role and responsibilities of unit testing in traditional development methods [3,11]. However, there is room for further research in the CSE world. This paper focuses on the very first step of the actual testing – viewing unit testing as a quality indicator during the development life cycle.

## The vague definition of unit test

Previous work [3] provided a detailed discussion of the term, “unit test,” argueing that there are two different ways to define the term:

* The classic way – About two thirds of the existing literature refers to a unit test as the smallest, isolated, atomic and code-related test that is performed mainly by software developers.
* The component way (24%) – The focus is on a unit of functionality, not necessarily on the perception of the unit test as the smallest, indivisible portion of the program; here, the unit testing is administered mainly by testers.

One could fail to assign a clear definition to specific usage terminologies, apart from the very obvious code attribute and the intended testers and intended candidates (i.e., the developers). However, once a the test is defined by an external tester, it might be detached from the code.

The work in [3] also differentiates the term, unit, from a component and sees an important distinction between unit testing, which refers to the use of xUnit testing, and component testing, which is more general and means the testing of only a portion of the program. Another distinction might stem from the abstraction level, in which case, the unit test will usually be affiliated with the code itself, and the component test may be expressed in functional or business terminology.

It may be wise to differentiate between the term, unit test, i.e. an action which presents the content of the item to be tested, and the actual application of unit testing, i.e. the process as an execution effort.

Unit testing may form the basis for component testing that can be considered a higher level of 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.

. However, 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.

## Unit testing in CSE context

A successfully passed test must continue to be administrated and pass as long as the codebase remains constant. Fulfilling ideal code conditions for unit testing includes isolation and atomic code [3] 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 dependency must be mocked. Mocking has shown itself to be a proven and effective technique and is a widely adopted practice [3]. 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.

Apparently, the definition of the term “unit test” is neither clear nor precise. Most of the literature considers the structural aspect of the term – e.g., atomic, isolated, and connecting the action to an X-unit testing infrastructure. About a quarter of the sources define the term more loosely and display a higher level of abstraction that does not restrict the definition and which allows an integrative portion of the program to be included in the unit being tested.

In light of the growing importance of the role of the level of unit testing, Chassidim et al., [3] recommend that two levels of testing be distinguished in the early stages of software development:

* **Unit testing** – This is the process of testing the isolated, atomic, and code-related portion of the software (a unit). The best candidates to perform this activity are the developers 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.

It is vital to seperate 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 [3]. The current study aims to explore common definitions, processes, and unsolved issues related to unit testing and its related activities in continious real-world environments.

## Continuous Integration (CI) and test automation

Continuous Integration (CI) is a practice where members of a team collaborate and integrate their work frequently; each developer integrates at least daily - leading to multiple integrations per day [10]. Each integration is verified by automated build, which includes testing to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly [10]. In CI development environments, software engineers frequently integrate new or revised code into the mainline codebase [12]. This approach can reduce the amount of rework that is needed in later phases of development and can speed up the overall development time using automated processes. Test automation is considered a critical prerequisite for continious development. However, its applicability and adaptation to testing are still limited.

Testing cannot be a time-focused operation but a continuous activity. Integration and testing are increasingly intertwined as software moves closer to deployment. Therefore, to be cost-effective, regression testing techniques must operate effectively within CI development environments [12]. Although, it has received very limited attention from the research community, CI, including compilation, building, and testing of software, is emerging as one of the success stories in automated software engineering [16,17].

## Testing levels in CSE

The Agile software development life cycle (SDLC) demands that all testing be done within a small development cycle. Thus, it is difficult to differentiate the testing activities within the Agile cycle, since they are done internally as part of the development routine and by the same people, who are not necessarily testers. This might be the reason for the unclear division between testing levels [18]. Other testing levels include User Acceptance Testing )UAT) and Non-Functinal Testing (NFT), conducted on production in more user-oriented environments. Figure 2 presents such a separation, portraying UAT and NFT as a separate testing stages prior to package delivery.

Figure 2. Possible testing levels in Agile SDLC

Prechelt et al., [19] dealt with the question of who is performing the Agile testing and evaluating the outcomes by performing a case study. They found that developers manage to fulfill the responsibilities of the conventional tester role by identifying which aspects can be covered by efficient automated testing or evaluated implicitly by end-users. The risk of the latter is minimized by the developers being able to react independently and quickly in case of problems, such that the benefits of having realistic and direct end-user feedback would prevail. In addition, they revealed that a prominent disadvantage of a “quality experience work mode” [19] appears to be that integration testing beyond the team level becomes harder.

Prechelt et al., [19] defined the “quality experience work mode” as quality assurance and deployment within a team (i.e., without dedicated testers). They argued that integration testing beyond the team level is more difficult, once it is performed by external testers who might hamper the feedback loop more than they contribute to quality. Quality experience and dedicated testers can probably be combined if friction can be avoided in the process. A limitation is that the above constraints cannot always be fulfilled. If development is too closely coupled with that of other teams or if deployment takes too long, a strong quality experience will not occur, and it may be difficult to change this. Company or team culture might also be affected. If teams are highly motivated and focused, development efforts may decrease due to less coordination overhead and the higher degree of automation that could compensate for the disadvantages.

## Continuous Deployment (CD)

Continuous Deployment takes CI practice one step further by automatically deploying software changes to production [9]. Continuous deployment emphasizes build and test automation together with a much-reduced scope for each release. Interviews with 15 information and communications technology companies revealed the benefits and obstacles to continuous deployment [9]. Despite understanding the benefits, none of the companies had adopted a fully automatic deployment pipeline. The study also reveals that adopting continuous deployment practices involves coordination from teams throughout the organization and the domain in which a company operates. Current SDLC presents the continuous nature of the cycle, where, after connecting the development into the release process, for example the DevOps approach [20,21], the monitoring action should provide feedback which initiates new planning actions for continuous development. An important aspect of testing within the Agile cycle is the necessity of test automation implementation as part of product testing. These automation artifacts will be used in a later regression package and will accompany the software in other usage contexts.

## Implementation of continious software engineering

In an attempt to model a benchmark for continuous integration implementation, [9] 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 [9], it is necessary to determine the advantages and disadvantages of the various aspects of CI. For this purpose, based on the findings in their study, they have proposed a descriptive model for better documentation of continuous integration variants. In previous work, [21] 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, and only four of them addressed software testing improvement. However, 39% of the studies mentioned that testing effort and time are critical factors. Not even a single study dealt with the effect of transitioning to CI and the corresponding implications of the quality and the testing processes. [22] focused on the question of visualizing end-to-end testing activities in order to support the transformation towards CI. With end-to-end testing, they refer to all code, from code written by individual engineers to product release. The aim of this research was to gain insights into how to support the transition towards continuous deployment in the software development industry. Their case studies proved some disturbing findings about the change processes: significant duplicate testing efforts, slow feedback loops, late testing of quality attributes, and 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 this research was the creation of the Continuous Integration Visualization Technique (CIViT) and the attempt to implement it for case study companies.

Recently, many organizations have adapted the Scaled Agile Framework (SAFe) and Disciplined Agile Delivery (DAD) in order to address the needs of larger projects [23]. Besides the development of code, the purpose of both frameworks is to take the architecture, project funding, and governance of the processes and roles required by management into account. At this level, the very same lean and agile principles that have worked well at the team level are applied. SAFe scales up the Agile method, Scrum. It focuses on the enterprise level, for example, organization-wide release planning sessions. Some research was done on the various roles within the Scale Agile Methods to Large Distributed Enterprises [24,25], but the quality aspects of the process have been largely ignored. 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, the unit test and testing related activities.

# Research methodology

## ****Participants****

Interviews were conducted using qualitative semi-structured interviews with open questions at the sites of three leading software development companies in Israel and USA (n=15). The interviewees were distributed per company, as shown in Table 1.

 

Table 1 Interviewees distributon

The companies had previously reported that they were already working in a continuous environment. The following are general descriptions of the companies:

Company 1, a large-size company, (5k+ developers) provides financial crime, risk, and compliance software solutions for international financial institutions and government regulators.

Company 2, a private ownership mid-size company (100+ developers), provides test and lab automation software solutions. The company serves network equipment manufacturers, service providers, data center operators, micro-electronics, and storage and electronics device manufacturers worldwide.

Company 3, an international large-size company (10k+ employees), provides infrastructure and maintainance services.

## ****Interviews****

To explore the actual practices of CSE, we took a general interview guide approach [26] in order to ensure that the same general areas of information were collected from each interviewee. However, in practice additional information was collected when interviewees raised other relevant topics based on their personal experiences and approaches right after answering the originally structured question. Figure 3 presents the process of preparation and implementation of the interviews.

First, we approached executive managers in charge of implementing the CI approach to confirm their company’s participation in the study and to ask for referrals to the relevant role holders to interview. Questions were sent to the managers for internal review and feedback (Fig 3. A). We then updated the questionnaire based on their input to improve the quality of the interviews. After consultation and approval from the top management of each company, we conducted the interviews with the relevant role holders (Fig 3. B). Before approval for publication, the top management applied an internal audit to the answers and provided supplemental materials to support the portrayal (Fig 3. C-D). To maintain the authenticity, the original terminologies and phrases were recorded and analyzed. The data were approved before publication (Fig 3. E).

 

Figure 3. Preparations for interview and information collection process

The following questions were presented to participants:

**Q1:** Prior to the implemetation of continious practices, what was the company approach regarding unit testing?

Insights from this question may shed light on the path taken by the company and the role of the starting point on the specific solution the organization adopted.

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

Answers to this question may bring understanding of their initial policy and methodology that may point to future difficulties and challenges with respect to the continious practices.

**Q2.2** What tools and technologies are used for unit testing?

**Q2.3** What tools are used for unit testing automation?

Since test automation is a key enabler of CI, the selection of a certain test automation technology may foresee the complexity of creation of the specific solution and future maintenance efforts.

**Q2.4** What are the other testing levels performed?

This identifies the organization testing scope.

**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 activities in the full context of process and product quality?

By looking at the full quality context, we may see the importance and impact of unit testing activities. This may open other research venues regarding implications of CSE projects to quality assurance.

In addition to these questions, the respondents described more issues and challenges they encountered in their company that were relevant to the context of this study.

## ****Data analysis****

Data were analyzed by compiling it into sections or groups of information, also known as themes or codes, in order to collect consistent phrases, expressions, and ideas that were common among research participants [27]. In order to minimize biases, this evaluation was applied seperately by two researchers of the study and also by a representative of each comapny.

# Case study findings

This section summarizes the responses and introduces insights derived from all the cases. All three companies participating in this survey had already adapted unit testing as a vital part of their development scheme. Morever, the unit testing role is intensifing and the CICD project positions the unit testing activities as central to all activities related to quality .

Generally, companies adapt unit testing automation as their main automation engine and define it using the classical definition (see Section ‎2.3) using X-unit as their main unit testing tool and environment. Test automation 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 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 test automation requirement 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 the measurements and the quality goals as their soft 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.

Unit testing has become the central quality assurance tool. In addition to the ease and availability to directly employ changes and enhancement, CICD unit testing has been described as the nearly the only way to produce validation and quality assurance.

**Issues raised following 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

- Unit testing 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 assembles its own tool selection and integrates all of them.

- The responsibility for quality is transferred to development teams.

- The measurements, tools, and standards are not mature.

- The economic benefits of the different testing levels has yet to be formalized.

- The industry is galloping forward without enough academic preliminary research. It is hard to see a theoretical justification and support for the new trends in the academic literature.

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, and the latter is developed and executed by testing experts. A possible explanation for this is that component testing is partially manual work and requires different test automation tools.

# Discussion and conclusions

Looking at CSE trends thru these case studies illustrates the importance of unit testing activities and outcomes for modern software development calling for explicit measurements and standards. The participating companies associate the unit testing activity with the image of quality. They all see it as a foundation for assuring the quality of their product. Nevertheless, they are still missing appropriate measurements and standards.

CICD trends affect the testing levels and eliminate the boundaries between different levels forcing a single unit to perform all the related activities. Therefore, it becomes vital to train and enrich the Agile team participants with unit testing proficiency and techniques. However, it is important to clearly distinguish between the different testing activities. Unit testing by itself is considered a milestones for achieving a better quality but does not deal with all the complexity of integration and functionally.

Agile development teams are required to perform new activities that were done by dedicated teams in the past. For example, a key success factor is the ability to automate all previously manual activities. In addition, new testing considerations should be addressed by the teams, including performance, security, deployability, and regression.

We have not seen a deep understanding for the long term planning, training, and organizational support for the change. It is possible that some of the testing and quality expertise might evaporate with the tendency to reduce the professionally oriented groups within the organization. Let us hope these organizations won’t dispense their previous knowledge and expertise before realizing the need. This case study might represent subjective opinions and therefore, we suggest approaching a greater number of software projects around the globe to validate and strengthen our findings.

We believe that the accelerating trend toward Agile software development will dramatically change the way unit testing is practiced. One of the main trends is that testers are embedded in the Agile development teams along with the programmers. The statement in our summary of the practices in Company 2, that unit testing “is done by the developers with the assistance of testing experts within the agile development teams” shows, we believe, that the model that will increasingly be followed by the vast majority of organizations moving towards Agile software development. There are several key advantages of this practice; historically, unit testing has been practiced by developers in isolation with widely varying consistency and effect. In Agile development teams, testers paired with programmers or at least advising programmers raise the level of consistency of unit testing and clearly improve the quality of the code produced. Since it is well-known that the earlier a defect is found in code, the exponentially less it will cost to fix, we beleve this is a very significant improvement in both software quality and software development cost control.

Traditionally, testers find defects in code and record as much information as they can in defect management systems. That information is then passed back to the programmers who then try to locate the source of the defects in the code. This can be a difficult and time-consuming task for the programmers. Clearly, testers working together with programmers in Agile teams can assist in finding the source of code defects. This is especially true if the testers involved have some level of familiarity with programming and with the programming language in use.

Another aspect, although not previously discussed, is the “exploratory testing,” known to improve the quality but 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. However, 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.

# References

[1] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” *IEEE Access*, vol. 5, no. Ci, pp. 3909–3943, 2017.

[2] D. Ståhl, J. B.-J. of S. and Software, and undefined 2014, “Modeling continuous integration practice differences in industry software development,” Elsevier.

[3] Chassidim, H., Almog, D., Sohacheski, D. B., Gillenson, M. L., Poston, R.& Mark, S.(2018). The Unit Test: Facing CICD- Are They Elusive Definitions? Journal of Information Technology Management, 29(2), 40.‏

 [4] “ISO - ISO 8402:1994 - Quality management and quality assurance — Vocabulary.” [Online]. Available: https://www.iso.org/standard/20115.html. [Accessed: 10-Mar-2020].

 [5] “ISO - ISO/IEC 25010:2011 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.” [Online]. Available: https://www.iso.org/standard/35733.html. [Accessed: 10-Mar-2020].

[6] H. Alaqail and S.Ahmed, “Overview of Software Testing Standard ISO/IEC/IEEE 29119,” IJCSNS Int. J. Comput. Sci. Netw. Secur., vol. 18, no. 2, pp. 112–116, 2018.

 [7] 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 Systems Assurance Engineering and Management, vol. 10, no. 5. Springer, pp. 867–897, 01-Oct-2019

[8] C. Condo and B. Seguin “The Forrester New WaveTM: Value Stream Management Tools, Q3 2018.” [Online]. Available: https://www.forrester.com/report/The+Forrester+New+Wave+ Value+Stream+Management+Tools+Q3+2018/-/E-RES141538. [Accessed: 09-Mar-2020].

[9] Olsson, H. H. and J. Bosch (2014). Climbing the “Stairway to Heaven”: evolving from agile development to continuous deployment of software. Continuous software engineering, Springer: 15-27.

[10] M. Fowler, M. F.- 16, 2018 from http://www. martinfowler. com/.... html], and undefined 2006, “Continuous integration. Thought-Works.”

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

[12] S. Elbaum, G. Rothermel, and J. Penix, “Techniques for improving regression testing in continuous integration development environments,” in Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2014, vol. 16-21-November-2014, pp. 235–245.

[13] J. Kasurinen, O. Taipale, and K. Smolander, “Software Test Automation in Practice: Empirical Observations,” Adv. Softw. Eng., vol. 2010, 2010.

[14] D. Kumar, K. M.-P. C. Science, and undefined 2016, “The Impacts of Test Automation on Software’s Cost, Quality and Time to Market,” Elsevier.

[15] K. Wiklund, S. Eldh, D. Sundmark, and K. Lundqvist, “Impediments for software test automation: A systematic literature review: Impediments for Software Test Automation,” Softw. Testing, Verif. Reliab., vol. 27, no. 8, p. e1639, 2017.

[16] M. Hilton, “Understanding and improving continuous integration,” in Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2016, vol. 13-18-November-2016, pp. 1066–1067.

[17] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, “Trade-offs in continuous integration: Assurance, security, and flexibility,” in Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2017, vol. Part F130154, pp. 197–207.

[18] R. Black, “Certified Tester Foundation Level Extension Syllabus Agile Tester,” 2014.

[19] L. Prechelt, H. Schmeisky, and F. Zieris, “Quality experience: A grounded theory of successful agile projects without dedicated testers,” Proc. - Int. Conf. Softw. Eng., vol. 14-22-May-, no. May, pp. 1017–1027, 2016.

[20] R. Pietrantuono, A. Bertolino, G. De Angelis, B. Miranda, and S. Russo, “Towards continuous software reliability testing in DevOPs,” in Proceedings - 2019 IEEE/ACM 14th International Workshop on Automation of Software Test, AST 2019, 2019, pp. 21–27

[21] M. Shahin, M. Ali Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” IEEE Access, vol. 5, no. Ci, pp. 3909–3943, 2017.

[22] A. Nilsson, J. Bosch, and C. Berger, “Visualizing testing activities to support continuous integration: A multiple case study,” in Lecture Notes in Business Information Processing, 2014, vol. 179, pp. 171–186.

[23] S. W. Ambler and M. Lines, “The disciplined agile process decision framework,” in Lecture Notes in Business Information Processing, 2016, vol. 238, pp. 3–14.

[24] J. M. Bass, “How Product Owner Teams Scale Agile Methods to Large Distributed Enterprises.”

[25] T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, “Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation,” Empir. Softw. Eng., vol. 23, no. 1, pp. 490–520, Feb. 2018.

[26] D. W. Turner, “Qualitative interview design: A practical guide for novice investigators,” Qual. Rep., vol. 15, no. 3, pp. 754–760, 2010.

[27] S. Kvale, “Interview Quality,” in Doing Interviews, SAGE Publications, Ltd, 2011, pp. 79–91.