Yüz Tanıma Tabanlı, Kısa Döngülü, Sürekli Entegrasyon ve Kalite Gözetimi*

Özen Özkaya[1]**, Hatice Erdoğan[1], Alphan Çamlı[1], Damla Gülen[1], Nihat Ük[1], and Tamer Temizer[1]

Siemens A.S., Istanbul 34870, Turkey
{ozen.ozkaya,hatice.erdogan,alphan.camli,damla.gulen,nihat.uk,tamer.temizer}@siemens.com

Abstract. Yazılım kalitesinin yüksek olması, organizasyonlar arası rekabette büyük öneme sahiptir. Teknoloji dünyasında yaşanan büyük değişimler göz önünde bulundurulduğunda, tüm bu değişimlere adapte olmak ve kaliteyi yüksek tutmak büyük önem arz etmektedir. Günümüz dünyasında, yazılım sürümlerinin sürekli entegrasyonu bu ihtiyacı adresleyen önemli bir pratik olarak öne çıkmaktadır. Sürekli entegrasyon ile yazılımın derlenmesi, test edilmesi, doğrulanması, dağıtımı ve kalite ölçümlerinin otomatik hale getirilmesi hedeflenmentedir. Kalite metriklerinin ölçümü ile, daha yüksek seviye kalite bilgilerinin, yazılımın her bir sürümü için takip edilmesi; yazılım kalitesinin artışına büyük katkı sağlamaktadır. Bu süreçte yaşanan genel bir problem ise,olası bir kalite düşüklüğünün çok geç farkedilmesi ve yazılımış önemli miktarda kodun tekrar değiştirilmesi ihtiyacının oluşması durumundur. Bu çalışma ile, sürekli entegrasyon sisteminin kısa döngülerle tetiklenmesine ilişkin, yüz tanıma bazlı bir yöntem önerilmektedir. Bu sayede geliştiriciye erken fazlarda kalite geri bildirimi verilebilmektedir.

Keywords: Çevik Yazılım Testi · Sürekli Entegrasyon · Test Otomasyon Çatısı · Güvenlik · Yüz Tanıma

* Supported by organization Siemens.
** Corresponding Author

Face Recognition Based Small-cycle Continuous Integration*

Özen Özkaya[1] **, Hatice Erdoğan[1], Alphan Çamlı[1], Damla Gülen[1], Nihat Ük[1], and Tamer Temizer[1]

Siemens A.S., Istanbul 34870, Turkey
{ozen.ozkaya,hatice.erdogan,alphan.camli,damla.gulen,nihat.uk,tamer.temizer}@siemens.com

Abstract. High quality software is essentially important in the giant competition field of organizations. When we consider into the fact that world of technology is changing rapidly, adapting all these changes and keeping quality high is a must to have in the competition. In todays world, continuous integration is a very good practice on addressing these needs. Continuous integration aims to automates building, testing, verification, deployment and quality assessment of the software on each release. Measuring code quality metrics and extracting out various software quality models support this organizations to have maintainable, bug-free and traceable software in the software development life cycle. As continuous integration automates all the steps mentioned, code quality is also tracked for each release; that results a higher code quality. The problem is that, it is mostly too late when you observe a low code quality on a release as a big amount of software needs to be refactorred or even rewritten. This paper presents a small cycle continuous integration and quality monitoring approach to give an early feedback to developer.

Keywords: Agile Software Testing · Continuous Integration · Test Automation Framework · Security · Face Recognition

* Supported by organization Siemens.
** Corresponding Author

1 Introduction

A typical software development process consists of specification, design, implementation, verification, validation, and maintenance. In literature, various software development process models are proposed such as waterfall [5], spiral [6], and agile [4]. Agile approach suggests to manage changes by verification and release of projects in small periods. To be able to manage this integration and release cycles wisely, an automated continuous integration system is essential. With this perspective, agile testing approach is mostly applied over continuous integration principle. Continuous integration helps to increase release cycle rate with more qualified software and more efficient team work [12]. Even it is obvious that establishing a continuous integration system has many advantages, it comes with hard challenges in practice. One of the most aspect in this challenge set is when and how to trigger continuous integration system in an optimal way. This question is not properly adressed in literature. In this study, we aimed to adress this issue with a new trigger approach.

In this paper, we describe an identification system with face recognition method within continuous integration approach. The system identifies developer during the time spent in front of the development computer (or working desk) and when developer leaves the computer, system detects it and after a while system triggers the continuous integration pipeline including code format checking, automated building, automated testing, quality measurement and dependency reporting. After developer comes back, the system identifies developer again and presents results to developer in a well organized report format. Even the face recognition algorithm itself is not the contribution of this study, it will be explained in further sections.

The main aim with this system that is proposed in this paper is to test at the earliest feasable time and use spare time of developer instead of allocating extra time for verification of software. In addition to this, in the long run, we target to motivate developers not to leave from their computer with incomplete development. Because incomplete code could not be fully tested, an error report is shown about that code. Empirical results showed that by using the proposed system developers tend to leave their computer with buildable code and ready for automated integration, instead of leaving in the middle of their work and getting error notification about incomplete integration and verification results. This tendency emprically resulted to a less number of bugs in the code.

2 Background

Philip B. Crosby who is one of the contributors of quality management has defined quality as conformance to requirements [14]. Crosby has adopted "Do it Right the First Time" and "Zero Defects" approaches [25]. He has proposed for the qualified system that requirements must be analyzed well and prevented errors instead of being detected later on. According to Crosby, "Zero Defect" is a performance method in regard to people deal with each details and aim to avoid

errors. In that way, people adopt zero defect goals [21]. Joseph M. Juran who is a missionary for quality management has defined quality with two critical definitions. First of them is "Quality" means to meet customer requirements and provide customer satisfaction, second definition is that "Quality" is related to "freedom from deficiencies" [1]. Juran has adopted quality planning, quality control and quality improvement approaches in order to manage quality. W. Edward Deming who is a quality expert has defined quality as the customer's current and future needs [25]. Deming's philosophy is to purify inconsistent parts of the system and provide defect free process with continuous improvement [21].

Agile methods along with continuous integration and test automation has shown that increasing software quality without compromising cycle times and development effort is possible [17]. In order to reach higher levels of quality, the agile methodologies are presented in 2001 [4]. Martin Fowler has described continuous integration as "Continuous Integration is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily leading to multiple integrations per day" [13]. In continuous integration system, testing should start as early as possible as agile methodology, when each change is integrated to system, automated tests run in parallel. Common practice in continuous integration is developers commit daily and build early. After a commit which is made into the mainline branch, a set of tests starts to run. The duration of the tests needs to be small enough to encourage developers to commit and build in small phases with smaller blocks of codes [19].

3 Related Works

One of the most essential software development practices of today is the continuous integration. Continuous integration is a well-established practice where development team integrates their work frequently. This integration is done by automated builds, automated tests, quality measurements, and further checks to detect integration errors, pitfalls or drawbacks as soon as possible [13, 10]. Continuous integration is reported to improve release frequency, predictability, developer productivity, communication and it is reported to reduce risks and defects [23, 15, 20, 9]. Continuous integration and agile testing are closely related that a continuous integration system is often considered as a key practice supporting agile development and testing environment [24].

Continuous integration systems generally include quality measurement and monitoring phase. So, measuring and monitoring software quality is as easy as checking the integration build results which shows the quality metrics and quality model calculations over the time [10]. This process is practically applied by using the tools like Jenkins, Atlassian Bamboo, and Teamcity [24]. In this paper, all of the studies are applied in the Jenkins environment.

Build frequency of continuous integration systems is a very important parameter in practice [23]. A very low frequency will break the continuity of the integration and a very high frequency will bring a huge integration overhead,

preventing team to advance. In literature, the major approach to determine frequency of continuous integration is mostly the release frequency. In most of the cases, continuous integration system is triggered on each check-in (or commit). The main idea is here to trigger continuous integration process when source code changes [18, 7, 22, 16, 8]. However, current practice on triggering continuous integration on each commit is not sufficient as most of the time a big amount of code is pushed out and furthermore, it becomes too late to restore a pitfall. Of course, triggering continuous integration each time code changes is not a good idea because it will create a huge integration overhead. This paper presents a face recognition based identification trigger for continuous integration systems to provide reasonably small cycle integration.

As the system uses face recognition based identification of developer (or tester), a real time face recognition system over webcam is developed and deployed on each development computer. A very good feature to describe facial features is local binary patterns [2]. Eigenfaces [26] and Fisherfaces [11] are also very popular methods which are used in face recognition systems.

Another complete solution on face recognition is Openface, which is a general purpose face recognition library which relies on deep learning [3]. As it has a proved success and it is easy to deploy, Openface can be used for face recognition in practice. In this study, there is no aim of developing a new face recognition approach; so any face recognition (or any other identification) system may be used instead. Local binary patterns based face recognition algorithm is implemented in this study which will be explained in the next section.

4 Face Recognition Based Small-cycle Continuous Integration

We present here our approach for providing a small cycle continuous integration system. The reason behind the need of a smaller integration cycle is that in most of the cases it becomes too late to redirect quality of the project with current integration cycle approach. Current approaches on literature mostly suggest to trigger continuous integration system once a code is released to repository. In this paper we suggest to trigger the continuous integration epoch by detecting the inexistence status of developer or tester.

System is deployed with a webcam which looks towards the face of the developer. In this process, system continuously captures the view of the developer in real time and it runs a face detection and then face recognition algorithm to identify if the developer is sitting across. No action is taken while developer is recognized by the system. Accordingly, while developer leaves the work place resulting being not recognized by the system for a while, system triggers the continuous integration process autonomously.

Once continuous integration process is triggered, system tries to build the local version of the code firstly. If the code is buildable, then artefact executables are used to run the automated tests. Even the code is buildable or not, quality metrics of the code is measured and higher level quality models are calculated

by using the lower level metrics. In the next step, documentation of the code and resulting reports of continuous integration system are auto-generated.

After continuous integration epoch is completed, a summary pop-up window is shown in the screen to inform the developer about the continuous integration results, once developer comes back to the working desk and re-identified by the webcam based face recognition system.

In this process, developer is informed on each break and quality is measured and monitored with a smaller-cycle. On each break, developer observes where quality goes, which modules are affected with the change and also the test results are shown if code is left buildable. In figures 1 and 2 you will find the overall diagram of the setup for the proposed face recognition based small cycle continuous integration system.

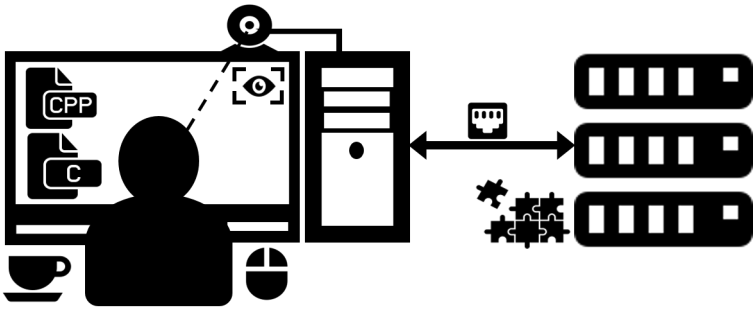

Fig. 1. Developer is existing and recognized by the system.

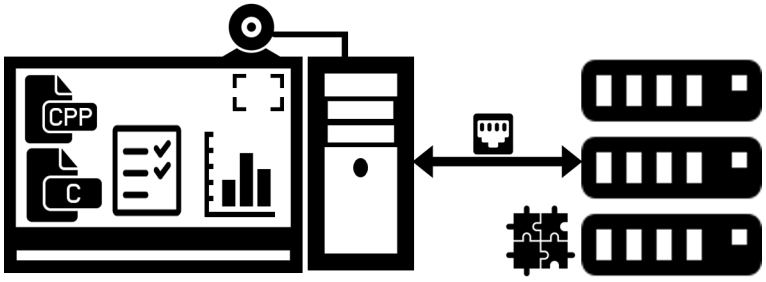

Fig. 2. Developer is not existing and not recognized by the system.

Process flow of the described system can be summarized in figure 3.

In the study revealed in this paper, face recognition is used as person identification as it is pervasive and easy to use but surely other identification methods like fingerprint, retina scan or even OS login may be used for the same purpose.

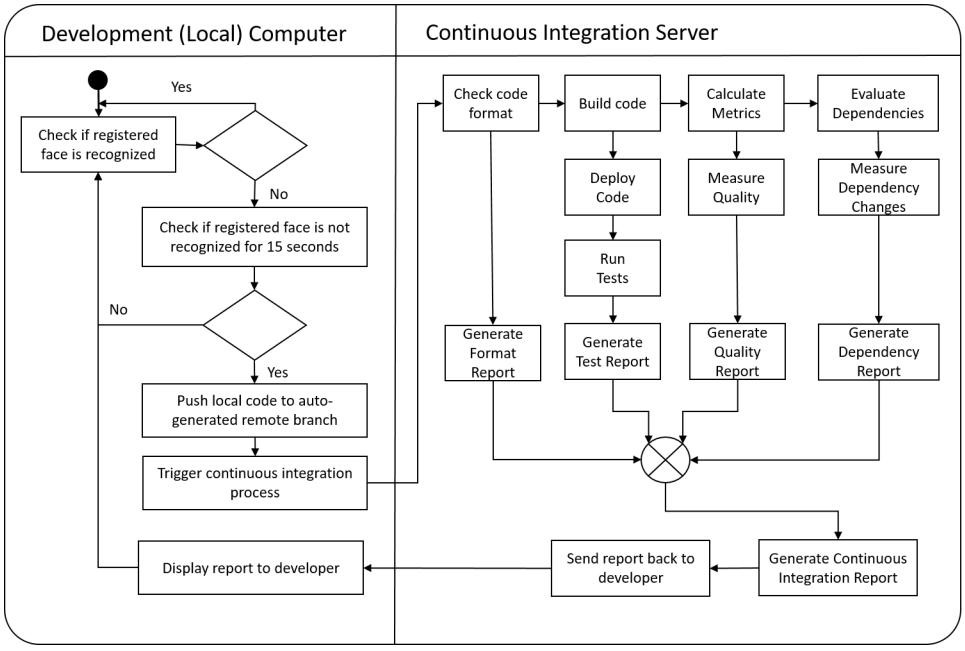

Fig. 3. Activity diagram of proposed system.

Since implementing a novel face recognition algorithm is not the one of the contributions of this study, a local binary analysis based face recognition application is developed. As the evaluations of the face recognition implementation resulted with %92 success ratio, it can be considered as highly succesful in recognition for the application purpose. OpenFace framework is also evaluated during the development but it is not preferred because of its dependencies to side packets. It is shown that OpenFace provides near-human accuracy on the LFW benchmark and it is always a choice for high precision recognition needs. [3]. In figure 4 you can find the architecture of the OpenFace framework.

As the continuous integration platform Jenkins is used in this study as it is highly customizable, easy to use, open source and it has a significant amount of plug-in support. Once trigger is arrived to continuous integration server, the first step is to check the code format. This process is done by using a formatter called "uncrustify" and comparing the properly formatted code with original code. If difference between the files produces an empty set, then format test is assumed to be passed. Otherwise difference file is added to format report.

The second step is to build the code. This step is usually achieved by using Makefiles. Once code is the compiled, other supporting artefacts like configuration files or databases are combined to deploy the system. Deployment test is a smoke test of the application which checks if vital feature like bring up works properly. Then all the automated tests including unit tests, feature tests and

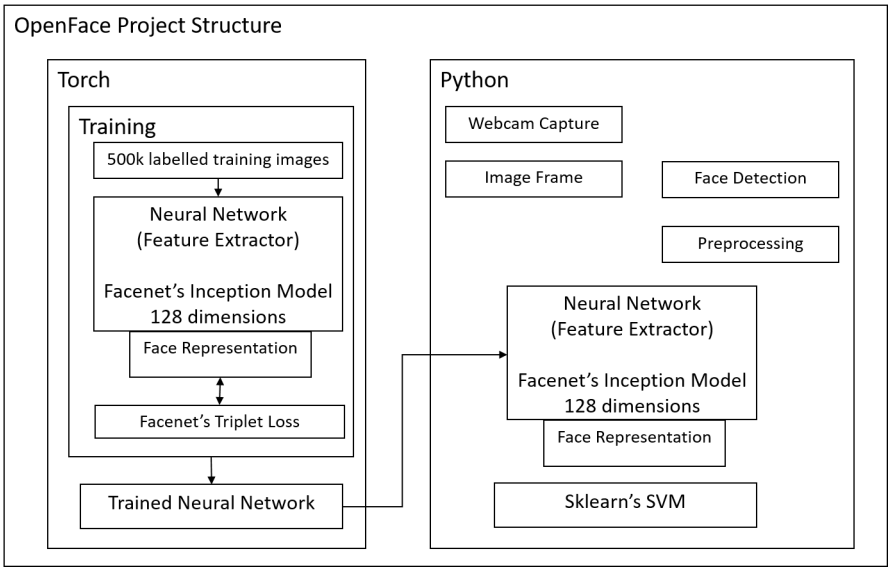

Fig. 4. Structure diagram of Openface.

memory usage tests are executed and reported. If code is not buildable, then test report only includes the information of build errors and warnings.

Even code is buildable or not, code quality analysis is executed. In this step, source code is statically scanned and basic quality metrics like McCabe complexity, QMin and duplication ratio are extracted for the project modules. This metric set can be extended according to the needs. In most of the cases, a higher quality model is constructed by using the basic quality metrics. At the end, a quality report is generated which includes not only the current quality measurements but also the change of the quality with previous release.

Similar to the quality measurement step, a dependency analysis is statically analysed for the source code version and newly created (or removed) dependencies are reported.

Finally, all the reports are attached to an e-mail and sent back to developer to inform the developer about the result of continuous integration cycle.

In practice, proposed system is built for embedded software projects while it can be used for any type of software development projects. The difference of embedded software is that codes are mostly written in C or C++ languages. Because of this fact, all the infrastructure is specially built to support C and C++ based projects.

Even it is possible to extend the quality model, in this study we used a basic quality model which will be common for both C and C++ projects. Because of these reasons, McCabe complexity, Qmin and duplication ratio are used as quality metrics.

5 Empirical Evaluation

We empirically evaluate the face recognition based small cycle continuous integration and quality monitoring approach by comparing it to traditional continuous integration approaches. Our experiments address the following research questions:

- RQ1: How it is better to pick epoch cycle in continuous integration systems?
- RQ2: What can be a useful trigger for continuous integration?
- RQ3: What works better to intervene quality when it goes worse?
- RQ4: What are the pros and cons of proposed system?

Empirical evaluation is executed for 4 project teams with 17 team members (developers) in total. Even it is hard to extract hard-numerical results, development teams made significant observations about the effects of the study.All the feedbacks are collected from the team members and quality of change is tracked for 2 weeks.

The first empirical result of the study addresses the first research question. If too large continuous integration cycle is preferred, then it would be too late for the corrections in the system and quality. In this scenario, agility of the team decreases sharply. On the other hand, when a too small continuous integration cycle is preferred, then continuous integration overhead come off dramatically which again decreases the agility of the team as development progress is slowed down by unnecessary continuous integration attempts. Both of too small and too large epoch cycle scenarios shows that just periodical trigger management is not a good solution. Empirical results are also verifying this conclusion.

Code commit based continuous integration trigger is usually hurts quality management in continuous integration process as each commit is done after a substantial amount of code is already written. There is no doubt in this case a smaller cycle continuous integration trigger approach is needed but the problem is that continuous integration overhead should not waste developer's productive time. In this stage, second research question emerges as a better trigger approach is essentially needed. Our approach which is proposed in this paper does not waste productive time of developers as it triggers the continuous integration system basically in developer's spare time. Our empirical evaluation of face recognition based continuous integration trigger approach is an optimal solution in terms of epoch cycle as all the process is executed when developer leaves the working desk, our solution creates zero integration overhead. In this manner, solution proposed in this study perfectly solves the research problem of picking an optimal epoch cycle in continuous integration systems.

Additionally, as face recognition based existence recognition system triggers the continuous integration process on developer's spare time, it is obvious that inexistence (of developer) based trigger is an optimal solution for continuous integration triggers. This empirical result leads to a clear answer to second research question.

Our empirical evaluations also showed that, face recognition based trigger triggers the continuous integration system 9.2 times a day on average. In this

process, continuous integration epoch is 52 minutes on average. This time is experimentally observed as the interval of implementation of a small software module. When developer leaves the place for a break up or a meeting, all the continuous integration pipeline is executed and reported back. As the implementation made in this interval is observed as a small software module, module's integration report is immediately reported back to developer when coming back to the desk. This optimal small cycle approach gave developer the opportunity to intervene quality when it goes worse. Hence, third research question is also answered with successful empirical outcomes. When comparing with previous month, quality measurements are boosted by %76 on average for all of the 4 projects.

As you can see from the figure 3, continuous integration system tries to build the application after code format check are done. In our empirical experiences, code is not always buildable when developer leaves the working desk. Even in this case, format checking, metric calculation over static code analysis, partial dependency analysis and related documentation is automatically executed by the system. Only run time tests and memory analysis are not done in this case as code is not left buildable, but all other steps are proceeding and developer informed with related reports.

After using the proposed system below, developers interestingly tend to left their desk with buildable code to be able to get more detailed reports and verification results of their implementation. After the first week of evaluation, %84 of the continuous integration triggers are executed on buildable code according to Jenkins logs. This tendency created fascinating outcomes as number of bugs per code line is sharply (%42) decreased when comparing with previous months. When trying to interpret the results, an important feedback is retrieved from the developers about a difference in their tendencies. After using the proposed system, a while, developers used to leave their desk with buildable code, it means a packed outcome of their studies and as their concentration is not broken during the block implementation, we observed fewer bugs in the projects. This outcome of the proposed system creates a huge impact on software development cycle and code quality. As each bug comes with additional development costs, our approach clearly decreases project costs while improving the quality sharply. This experiences answers the forth research question; even unbuildable code integration attempt can be seen as a con of the system, changing developer's tendencies to leave their desk with buildable code created a great positive impact.

One of the cons of the system occurs as false positive triggers. In some cases, developers sits in front of computers but does not actively write codes. Documentation or high level design tasks are good example of such cases.

6 Future Work

In this process, small CI phases can as early as possible to give result. And each breaks will share CI result instead of waste of time. CI parts can be extended with developers' behaviour. Because day by day each developers will be learned to

leave good, clean code as buildable, executable, runnable before giving a break. And this behaviour will get into habit of doing for developers. And with this habit CI will be more useful for software with early feed-back to developers for; improving Code Quality, developing the traceable software, unit test, etc. We can also improve small CI cycles even adding smoke Tests and getting its result too. So, we can observe the test phase, if development sends package to test, test can start or not?

7 Conclusions

In the software world, each day is coming with new steps and innovations. All of us who has a role in this world should catch the innovations and one step forward. And if we combine new ideas with the new features which are coming with new technologies, we can be a part of the future creation team of the software world.

In this paper, we focused on Continuous Integration part for building the automation, testing, verification, deployment, and quality of the software. But nowadays, analyzing the quality is only possible at the end of the CI (Continuous Integration) for each releases. In the aproach revealed in this paper is written for small CI cycles based on face detection of the developers. In each break of the developers system will be triggered with face recognition automatically for CI, and results will be shared with them when they will sit in front of the computer again.

For reaching the result in this paper face recognition method is described for triggering the CI. And it shows each break is appraisable and it can be returned with a result.

References

1. AB, J.J.G.: Juran's quality handbook (1998)
2. Ahonen, T., Hadid, A., Pietikainen, M.: Face description with local binary patterns: Application to face recognition. IEEE transactions on pattern analysis and machine intelligence **28**(12), 2037–2041 (2006)
3. Amos, B., Ludwiczuk, B., Satyanarayanan, M.: Openface: A general-purpose face recognition library with mobile applications. CMU School of Computer Science (2016)
4. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al.: Manifesto for agile software development (2001)
5. Benington, H.D.: Production of large computer programs. Annals of the History of Computing **5**(4), 350–361 (1983)
6. Boehm, B.W.: A spiral model of software development and enhancement. Computer **21**(5), 61–72 (1988)
7. Bowyer, J., Hughes, J.: Assessing undergraduate experience of continuous integration and test-driven development. In: Proceedings of the 28th international conference on Software engineering. pp. 691–694. ACM (2006)

8. Dösinger, S., Mordinyi, R., Biffl, S.: Communicating continuous integration servers for increasing effectiveness of automated testing. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 374–377. ACM (2012)
9. Downs, J., Plimmer, B., Hosking, J.G.: Ambient awareness of build status in collocated software teams. In: Proceedings of the 34th International Conference on Software Engineering. pp. 507–517. IEEE Press (2012)
10. Duvall, P.M., Matyas, S., Glover, A.: Continuous integration: improving software quality and reducing risk. Pearson Education (2007)
11. Fisher, R.: The use of multiple measures in taxonomic problems. Ann. Eugenics. v7 pp. 179–188
12. Fitzgerald, B., Stol, K.J.: Continuous software engineering: A roadmap and agenda. Journal of Systems and Software **123**, 176–189 (2017)
13. Fowler, M., Foemmel, M.: Continuous integration. Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf **122**, 14 (2006)
14. Galin, D.: Software quality assurance: from theory to implementation. Pearson Education India (2004)
15. Goodman, D., Elbaz, M.: " it's not the pants, it's the people in the pants" learnings from the gap agile transformation what worked, how we did it, and what still puzzles us. In: Agile, 2008. AGILE'08. Conference. pp. 112–115. IEEE (2008)
16. Janus, A., Schmietendorf, A., Dumke, R., Jäger, J.: The 3c approach for agile quality assurance. In: Proceedings of the 3rd International Workshop on Emerging Trends in Software Metrics. pp. 9–13. IEEE Press (2012)
17. Jeffries, R., Anderson, A., Hendrickson, C.: Extreme programming installed. Addison-Wesley Professional (2001)
18. Liu, H., Li, Z., Zhu, J., Tan, H., Huang, H.: A unified test framework for continuous integration testing of soa solutions. In: Web Services, 2009. ICWS 2009. IEEE International Conference on. pp. 880–887. IEEE (2009)
19. Meyer, M.: Continuous integration and its tools. IEEE software **31**(3), 14–16 (2014)
20. Miller, A.: A hundred days of continuous integration. In: Agile, 2008. AGILE'08. Conference. pp. 289–293. IEEE (2008)
21. Monnappa, A.: Pioneers of project management: Deming vs juran vs crosby (2017), https://www.simplilearn.com/deming-vs-juran-vs-crosby-comparison-article
22. Pesola, J.P., Tanner, H., Eskeli, J., Parviainen, P., Bendas, D.: Integrating early v&v support to a gse tool integration platform. In: Global Software Engineering Workshop (ICGSEW), 2011 Sixth IEEE International Conference on. pp. 95–101. IEEE (2011)
23. Ståhl, D., Bosch, J.: Modeling continuous integration practice differences in industry software development. Journal of Systems and Software **87**, 48–59 (2014)
24. Stolberg, S.: Enabling agile testing through continuous integration. In: Agile Conference, 2009. AGILE'09. pp. 369–374. IEEE (2009)
25. Suarez, J.G.: Three experts on quality management: Philip b. crosby, w. edwards deming, joseph m. juran. Tech. rep., TOTAL QUALITY LEADERSHIP OFFICE ARLINGTON VA (1992)
26. Turk, M.A., Pentland, A.P.: Face recognition using eigenfaces. In: Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on. pp. 586–591. IEEE (1991)