# Software Testing: A Changing Career

Sean Cunningham[1], Jemil Gambo[1], Aidan Lawless[1], Declan Moore[1], Murat Yilmaz[2], Paul M. Clarke [1,3] [0000-0002-4487-627X], Rory V. O'Connor[1,3] [0000-0001-9253-0313]

[1] School of Computing, Dublin City University, Ireland
{sean.cunningham29,jemil.gambo2,aidan.lawless8,
declan.moore39}@mail.dcu.ie
[2] Department of Computer Engineering, Cankaya University, Turkey
myilmaz@cankaya.edu.tr
[3] Lero, the Irish Software Engineering Research Center
{rory.oconnor,paul.m.clarke}@dcu.ie

The software tester is an imperative component to quality software development. Their role has transformed over the last half a century and volumes of work have documented various approaches, methods, and skillsets to be used in that time. Software projects have gone from using monolithic architectures and heavy-weight methodologies, to service-oriented and lightweight. Testing has transformed from a sequential step performed by dedicated testers to a continuous activity carried out by various development professionals. Technological advancements have pushed automation into routine test tasks permitting a change of focus for the tester. Management styles and methodologies have pushed development to be agile and lean, towards continuous integration and frequent release. Regardless of these many important changes, the software tester's role remains the verification and validation of software code.

**Keywords:** software quality improvement, test drive development, continuous development, software development lifecycle

## 1 Introduction

It is well documented that software development is a complex socio-technical activity [42], the process for which needs to be vigilantly maintained and evolved [43]. We furthermore find that roles within software development can have varying names [44], and that the very terminology adopted is perhaps the source of some fusion [45] [46]. We have also seen that the process and the details of work carried out when producing software are also variable given any particular development setting [47] [48]. Irrespective of these many important concerns, as a fundamental activity, successful commercial software development must incorporate a testing function. The precise role of the tester may vary in different organisations and change over time, and in this paper, we examine the nature of this change.

The role of a software tester has not fundamentally changed in the sense that software testers verify and validate that software complies with requirements. However,

what has changed are the tools and practices used to achieve this, and this affects both traditional and contemporary development methodologies. Recent changes in the practices and tools used by the software tester are a result of the increased utility of faster, better, and cheaper computing; which has permitted automation as a core component of modern test suites. Despite the evolution of the methods and tools of a software tester, the role at its foundation remains unchanged. The tester has adjusted the testing to suit the modern environment of software development in order to achieve the same outcome.

In this paper, we examine the impact on the role of the software tester arising from changes in software development lifecycle models, carefully examining the specific impact of agile methodologies and DevOps based software development. We also examine the changing nature of software projects themselves over time and identify how automation has had a lasting and significant impact on the role of a software tester.

## 2 Related Literature

### 2.1 Methodology

The method used for finding appropriate papers involved a set of search strings on academic search engines which include ScienceDirect, ACM Digital Library, IEEE Xplore, and Wiley Online Library. Search strings used included and combined: “software testing”, “role software tester”, “agile software testing”, “heavyweight software methodologies”, “software methodology changes”, “future software testing”, “function as a service testing”, “software test automation”, and “manual automated software testing”. Software quality focused publications required more specific strings, whereas general software journals produced relevant papers on general software testing strings.

Many search strings were generated through iterations of a snowball approach to the literature found. Recurring themes and elements of relevant literature would produce new search strings which could be used to generate further relevant sources. This snowball approach is also applicable to additional paper discovery. Relevant literature would cite sources for its content which in turn provided additional sources for our process.

While this paper seeks to primarily source any statements from academic, peer-reviewed sources, there is a clear value to utilizing multivocal sources. Particularly in the discipline of computer science, where there is a diverse and expansive online community discussing the role of the software tester. As such we have included analyses from certain non-peer reviewed sources.

Academic textbooks on the topic of software testing and software development have too proved useful in literature discovery. These texts reference academically reliable sources which can be also gathered to strengthen statements of this work.

### 2.2 Inclusion/Exclusion Criteria

Any of the sources this paper uses must be available in full-text; either electronically or in print. All papers should also be in English. While we are approaching this research

paper from a multivocal point-of-view, the merit of a peer-reviewed paper will supersede a non-peer review document.

Discovered papers, beyond the title relevance should present further potential as a relevant source to the topic. Each paper with relevant title was briefly assessed for additional significance and compiled into a list of relevant papers along with a brief summary for later.

Papers found through the methodology outlined above are given additional merit and considered more likely for inclusion if they are from the following journals; or where the authors are otherwise reputable contributors to these publications: Transactions on Software Engineering; Information & Software Technology; Journal of Systems & Software; Software Testing, Verification & Reliability; Transactions on Software Engineering & Methodology; Software Quality Journal; IEEE Software; Journal of Software: Evolution & Process.

Exclusion criteria on papers discovered included documented specific, non-industry standard, approaches or applications of specific tools. These papers, while loosely relevant to the topic don't typically provide much insight into general topics but more theoretical assessments of their subject's validity.

### 2.3 Papers Discovered

**Table 1.** Papers discovered using methodology outlined above

| # of papers | TSE | IST | JSS | STVR | ToSEM | SQJ | IEEES | JSEP |
|---|---|---|---|---|---|---|---|---|
| Identified | 29 | 29 | 25 | 11 | 6 | 22 | 19 | 6 |
| Included in analysis | 3 | 3 | 1 | 1 | 0 | 4 | 2 | 0 |

## 3 Analysis

### 3.1 Definition

Definitively, software testing can be traced back to 1957 when the differentiation between code debugging and program testing was established by C L Baker [1].The role developed from then in various directions. However, few were formalised until Myers book in 1979 [2] which discussed how despite 50% of software project time being spent on testing, that less is known about it than any other component of SDLC.

The traditional role of the software tester can be defined as one responsible for creating test plans and testing the developed programs [3]. From a philosophical approach, the software tester is responsible for verification and validation of developed software.

### 3.2 Software Development Lifecycle Models

**Traditional Methodologies**

While many would assume that the practices of Incremental and Iterative Development (IID) are recent concepts evolving from the surge in agile methodologies, they can be seen throughout the history of software development [4]. Practices like those employed in Extreme Programming are evident as far back as NASA's Project Mercury (1958), which used test-first development, which included time-boxed, half-day iterations.

Despite early evidence of IID, many software developers in the 1960s used the ad-hoc 'code-and-fix' method. This method has been described as *"one year of slamming code, one year of debugging"* [5]. This flawed method was the worst-of-both worlds for the software tester. It provided little in defining test parameters, didn't improve process quality, nor did it assist in optimising the economics of software testing.

The 1970s heralded the 'Waterfall' method, which was introduced by Winston Royce [6]. This methodology advocates a logical sequence of phases to be followed. The completion of the previous phase must be complete before continuing to the next. However, common misconception of Royce's method views the entire process as a single-pass through the phases, whereas Royce suggests the process be carried out across two iterations [4]. For the software tester, their role is fixed to the later phases of the life-cycle. As such, testing tended to be viewed as a dynamic testing activity that commenced once workable software was available from the programming team.

The V-Model approach to software development views development and testing as corresponding activities of equal importance. This model attempts to validate and verify the development based on levels of abstraction of the system which are sequentially tested throughout the process. This model distinguishes the objectives of each test level, which provides greater definition to the software tester's role in terms of the skills and tools required for each step. [7]

**Agile Methodologies**

As discussed, IID practices have existed long prior to the modern lexicon of agile methods. Caroline Wong assessed a 'build-approach' in 1984 applied to a critical systems development project, resulting in "*an on-time and within budget delivery of a high quality product that meets all operational requirements*", an early comparison to a traditional 'phase-approach' model. [8] This approach shows an improvement to measurement and control of the project, software continually tested and retested, and better adherence to requirements- all of which provide a more refined definition to the software tester's role, at least when compared to earlier methodologies. Similarly, Gilb's Evolutionary Development proposal results in a comprehensive documentation of the role of testing within the SDLC. This method shows the continuous development principles in their early stages, with greater consideration for the role of the tester with the SDLC. [9]

The introduction of Scrum to software development brought with it several hypotheses which challenged the Waterfall model. The Waterfall model wasn't flexible in circumstances of defects that are best remedied through altering requirements as opposed to the code, it was never designed for retrospective activities [10]. Unfortunately, this has meant that defects within a single phase are difficult to guarantee resolution for, thus the rigidity of the model can cause mutation of defects as the project progresses.

Scrum differs from Waterfall in its definitions of roles, meetings, characteristics, artefacts, and language [11]. While the software tester's fundamental role within Scrum remains unchanged, scrum assists the role by permitting team contributions to all phases. The team are encouraged to informally and frequently report on; what they have covered, what they plan to do next, and what issues or blockers they have experienced. Using this structure, testers play a more active role in development and members of the team can more effectively communicate resulting in a lower degree of coupling [10].

Scrum's role structure avoids assigning strict singular roles to team members, like automation engineer or manual tester, instead scrum requires testers to work across the full stack of quality assurance skill sets [12]. These individuals should be subject matter experts in the area of testing, while having a proficiency in a technical language set. The testers role under scrum has evolved to a more generalized approach. The effect this change in definition has to the role is apparent in the ambiguity of job perception. [13]

Within Agile methodologies [14] a shift in the tester's role can also be observed. Under many applications of Agile, testing is undertaken by most members of the team in some fashion [15]. This can include developers performing functional testing and requirement verification on their own code, and product owners performing usability testing. Additionally, Agile often emphasizes the need for User Acceptance Testing as a core component to its QA process [16]. Distributing the quality improvement tasks across the team allows quality assurance engineers to focus on value added testing, which include edge case, negative, integration, and exploratory testing [12]. The addition of individuals performing testing can increase the quality of the software, including defect detection and efficiency in resolving defects [17]. The reduction in workload permits QA to work on non-functional testing like performance testing, accessibility testing, security testing, and load testing- all of which become increasingly important in the shift to cloud based, service-oriented architectures.

Extreme Programming (XP) is a software development method, aligned closely to Agile methodologies. It focuses on applying good programming techniques, clear communication and teamwork to address issues that arise within lengthy development lifecycles and traditional methodologies [18]. It also emphasises eliminating duplicates from code and in terms of testing suggests users *"don't write a line of new code unless you first have a failing automated test."*

Testing in XP, expects developers and other non-test-dedicated team members to take responsibility for quality improvement tasks. A dedicated tester is not typically defined within XP and the role is performed by the customer and developer. Test Driven Development (TDD) arises from XP and shifts the focus of testing roles towards acceptance, integration, and system testing; unit tests and regression bugs are dealt with through the development methodology and arising defects are handled earlier in the process.

**DevOps and modern software development**

DevOps bridges the gap between development and operations using communication and collaboration, continuous integration, quality assurance and delivery with automated deployment. It is often compared with Agile principles and practices as they both

have similar goals and values but vary in scope [19] [20]. Modern software is often heavily partitioned across a set of interconnected, high availability services, with frequent deliveries. The quality of software in this context requires thorough and constant monitoring to avoid issues [21]. For a DevOps team to effectively handle changes in their production environments, the methodology is built on the core principles of: Emphasis on collaboration between development and operations; and Use of Agile principles and automation to configure and manage deployment environments.

The software tester's role within DevOps is to continuously test in order to identify defects as quickly as possible. The time savings of this testing approach permits exploratory testing and other value-added test activities to take place. DevOps prioritises test design, test case development, and automated testing within the tester's role. Testers should be familiar with a suite of ever-changing test tools and processes for continuous quality improvement.

With evolutions in cloud computing around how applications are deployed, we see the rise of Function as a Service (FaaS), serverless computing via serverless architecture. FaaS promises an alternative to PaaS due to its resource efficiencies [22]. Testing for FaaS applications might primarily involve unit testing and integration testing on smaller pieces of code and how each function integrates. This testing is reliant on externally provided systems. With the advent of these methodologies and architectures, testers are continuously adapting their skill sets. Despite these changes we still observe that the tester is there to verify and validate that code is written and runs correctly.

### 3.3 Changing Projects

Over the past 50 years software projects have evolved due to *"increased complexity size and diversity of demand"* [23] which has seen a shift from traditional methodologies to agile and lean approaches. The use of either traditional or agile methodologies depends on the speed of completion size of the system and level of collaboration between the development team members. Projects which need a simple working prototype developed quickly and later evolve by adding more features would use the agile methodologies. While traditional methods are better suited for projects with clearly defined specifications and requirements, particularly high criticality software as used in healthcare systems or aviation industry applications [5].

As methodologies change, so too do the people who use them. Older more mature models give strict descriptions as to the roles and responsibilities of a tester. In the Waterfall model, testing is performed at the verification phase using a documented set of tasks previously discussed. In the V-Shaped model, the tester performs testing for each phase of the cycle. These models apply to projects with rigid and well understood requirements.

Modern projects are often software for consumers or web applications and typically have a volatile set of requirements. To accommodate the dynamic need of these projects, structured methodologies like Agile are adopted. However, the role of the tester varies across these methods. Agile for example, offers vague guidance on testing whereas, XP testing is performed by everyone, with a focus on acceptance testing and

skill transfer. Scrum requires testers to have skills in programming in order to prioritise value-added testing such as negative, edge-case, integration, and exploratory testing.

### 3.4 Automated Processes

An aspect of the software testers role which has changed and continues to change is test automation. Test automation involves a task with the testing process which at one time might have been done manually by a human and automating it using applications and scripts. Automation can refer to automating the execution of test scripts which were written by a human or go so far as to automate the generation of these test scripts, execute them, and take actions based on the result. Machine learning has permitted automated testing to extend to defect detection, including dynamic models and defect predictions, further reducing the manual roles of software testers [24].

Software testing has increasingly adopted automated process executives over traditional manual testing for numerous reasons, including economic [25], efficiency, and accuracy improvements to modern test processes.

For manual testing, testers would write each test case and manually execute each sequentially, much like the end user of the application through routine use. Manual testing may involve exploratory testing, which doesn't rely on formal test case definitions. Rather, testers design tests based on their experience and intuition and execute them on the application. The issue with this method of testing is the rapid accumulation of test debt [26]. Test debt, like technical debt refers to unchecked issues within code that are not addressed in testing. Test debt occurs due to inadequate test coverage- informal exploratory testing results in inconsistencies to coverage. By automating test case execution, software testers can focus on maintaining test scripts and increasing coverage of testing.

Automated test execution eliminates many human errors resulting from manual testing. Mundane and repetitive tasks, such as logging into a system or visually observing incorrect code, are prone to human error [27]. Automating tasks like these save time and money and enable continuous delivery of a product. As a result, modern software development views test automation as an integral component [28].

Despite benefits to automated testing over traditional manual methods, the primary driver for testing disciplines adopting automation is as a result of changes to software life cycles. As noted by Mariani et al. [28] *"The philosophical principle of "test driven" processes in agile methodologies and more generally the 'shift-to-the-left' principle of DevOps have put test automation at the heart of software development processes."* These time-sensitive methodologies requiring continuous delivery of software, which leads to a need to effectively time-box activities like testing, reducing the level of experimental manual testing possible. The rapid growth in use of continuous software development practices like Lean and Agile has made automation, or the path to automation standard procedure of software testing.

### 3.5 Testing Approaches

Since Charles L. Baker differentiated between code debugging and program testing in 1957 [1], software testing in practice has evolved considerably. Ten years later IBM released a report which advocated a disciplined approach to software testing [29]. A year after that, NATO highlighted the importance of software quality assurance [30].

In 1971 mutation testing entered the lexicon of testing. Mutation testing involves using several modifications of an original program which contain 'artificial changes' and executing test cases on them [31]. Mutation testing is particularly useful for evaluating test suite quality, however it is time consuming and costly and as a result has fallen out of favour with modern day practices. Many testing approaches that were proposed around the same time and that have stayed prevalent in testing practices. These include structured programming, 'assertion statements for program proof', and 'top-down' testing [32].

Software testing techniques continued to evolve to suit the needs of a project's requirements, scope and resources. The economics of testing has always driven the approaches and methods of testing implemented [33]. It's understood that the cost of fixing a defect increases exponentially as time passes. Approaches like black-box testing were developed to assess the functionality of a program without foreknowledge of its implementation [34]. Later white-box testing was developed to improve the code structure, as opposed to focussing on functionalities; becoming a mainstay of data-flow testing [35]. The concept of 'walkthroughs' were introduced around the same time [36] and are analogous to proofreading software. Informal code reviews have consistently been argued to bring added value through tacit human knowledge to quality assurance [37], all the while the debate on 'who should test?' continues [38] [15]. Some believe developers should test their own code, while others view dedicated testers as necessary. Across the industry, teams apply combinations of these approaches and philosophies.

The 1980s saw further formalisation of testing practice. Following Myer's, The Art of Software Testing [2], the British Standards Institution published the Quality Assurance Standard which later merged into ISO 9000. IEEE 829 was published in 1983 and was a standard for test documentation. These developments are milestones in the change of the software testing process towards a more formal and efficient consideration.

Exploratory testing grew in popularity and emphasised personal freedom for the tester [39]. It allowed a software tester to design, execute, and interpret test cases simultaneously and use their knowledge and experience to discover more defects than structured testing might. Like other approaches, exploratory testing stands the test of time and remains in use for modern testers [40].

In 2001 when the Agile Manifesto was published [14], there was a widespread push for 'continuous integration' – bringing teams' work together more frequently; and 'continuous deployment' - deploying features of systems more frequently. The approach to software testing became about testing smaller modules much more frequently. Extreme Programming (XP) and Test Driven Developed (TDD) followed later as applications of the manifesto [18]. XP utilised pair-programming a relied on all developing team members to take part in the quality improvement process. TDD advocated that code be

written with testing in mind and remains a philosophy in many agile teams today. TDD principles can be seen in the work of Hetzel and Gelperin in 1988 [41] who note that testing is in a "prevention-oriented period", where tests were to demonstrate that software satisfies its specification, to detect faults and to prevent faults.

The software tester's role should continue to involve common methods. We suspect that domain specific techniques will accompany these. More recent technological progress such as blockchain, IoT, edge computing, and machine learning will all require the development of new approaches.

## 4 Research Limitations & Future Work

This research was initially undertaken by a team of four final year computer science undergraduates. Given their lack of research experience, it is possible that certain aspects of academic rigour may present with shortcomings in this work. To offset this risk, the students were provided with intensive research training at the outset and had weekly engagements with their research supervisor to discuss progress and techniques. The students were further constrained by a 6 week research window in which the topic was to be extensively and systematically investigated and written up. An additional limitation experienced related to the quality of various sources. Many papers discovered were of considerable age and were speculative of long surpassed future trends. In an industry like technology, frequent innovations prove many such sources obsolete. This gave insight into the value of many contemporary speculative papers and the conclusions they draw. Often multi-vocal, industry produced sources contained valuable discussion into the changing role of the software tester. However, the validity of these texts was diminished compared to that of peer-reviewed, academic sources.

It is furthermore the case that the scope of this research was not sufficient to identify certain additional (or perhaps emerging) concerns for contemporary software testers – for example, there is no explicit discussion of AI, deep learning and data driven development concerns. Hardware-in-the-Loop (HIL) environments for embedded systems and test case generators are also outside the scope of the work and would be interesting to examine in future extended research in this space.

Future research might examine specific industrial environments for variation in software testing and to undertake an in-depth analysis of the scope of a software tester's role and responsibilities in practice. It would also be interesting to produce a roadmap for evolution of the Software Tester role.

## 5 Conclusion

Despite half a century, the role of the software tester remains as vague as it was in Myer's observations in 1979. The migration to more agile development practices or more generalist team role definitions are in response to the changes in projects and technology. Software testing remains the practice of verifying and validating software, despite the change in tools and methods. Speculating on the future of the role of the software tester remains a challenge, and the evidence that historical speculation on such

changes being inaccurate supports this. We suspect more domain-specific approaches to existing techniques and tactics will be needed to correspond to recent innovations in IoT, edge computing, machine learning and blockchain technologies.

We have also seen that as software development methods and techniques have evolved, so too has the day to day work of software testers. Testers are increasingly required to automate substantial parts of their work, and automation is quite a different skill to classical software testing. Our research has also shown that over time, testers have become less distant from the core development (and operations) teams, where multidisciplinary members constitute cross-functional Scrum teams. One further parallel development has been the impact of testers on early testing: since testers form part of the Scrum team, they can now bring a testing mindset across the development lifecycle (and not just test executable code once it becomes available).

In conclusion, we have found that the fundamental role of the tester has not changed: testers are responsible for the validation of software. However, there have been many significant changes to the way that testers fulfil their role. Testers are now working much more closely with developers, operations engineers and even client representatives, and they are increasingly adopting automation technologies to replace what where originally manual tasks. Whereas in the past, a software tester may have forged a career running manual tests over and over again, this type of testing is in decline, testers increasingly need to be more technically savvy, able to integrate automation technologies into their daily tasks.

**Acknowledgement.** This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 to Lero - the Irish Software Research Centre (www.lero.ie).

## References

1. C. L. Baker, “Digital Computer Programming,” Mathematical Tables and Other Aids to Computation, vol. 1, no. 60, pp. 298-305, 1957.
2. G. J. Myers, The Art of Software Testing, Hoboken, New Jersey: John Wiley & Sons, Inc., 1979.
3. M. Yilmaz, R. O'Connor and P. Clarke, “A systematic approach to the comparison of roles in the software development processes,” in 12th International Conference on Software Process Improvement and Capability Determination, Palma, Spain, 2012.
4. C. Larman and V. R. Basili, “Iterative and Incremental Development: A Brief History,” IEEE Computer, vol. 36, no. 6, pp. 47-56, 2003.
5. M. A. Awad, “A Comparison between Agile and Traditional Software Development Methodologies,” The University of Western Australia, Perth, 2005.
6. W. W. Royce, “Managing the development of large software systems: concepts and techniques,” in Proceedings of the 9th international conference on Software Engineering, IEEE Computer Society Press, Monterey, California, 1987.
7. A. Spillner, T. Linz and H. Schaefer, “The General V-Model,” in Software Testing Foundations, Santa Barbara, California, Rocky Nook, 2014, pp. 39-42.
8. C. Wong, “A Successful Software Development,” IEEE Transactions on Software Engineering, vol. 10, no. 6, pp. 714-727, 1984.
9. T. Gilb, “Evolutionary Delivery versuse the "waterfall model",” ACM SIGSOFT Engineering Notes, vol. 10, no. 3, pp. 49-81, 1985.

10. M. Mahalakshmi and M. Sundararajan, “Tradition SDLC Vs Scrum Methodology - A Comparative Study,” Internation Journal of Emerging Technology and Advanced Engineering, vol. 3, no. 6, pp. 192-196, 2013.
11. K. Schwaber, “SCRUM Development Process,” in Business Object Design and Implementation, London, 1997.
12. B. Boemmel, “The Changing Role of Software Testing in an Agile Environment,” 17 March 2017. [Online]. Available: https://softwaretesting.cioreview.com/cxoinsight/the-changing-role-of-software-testing-in-an-agile-environment--nid-23976-cid-112.html. [Accessed 20 February 2019].
13. W. Sun and C. Schmidt, “Practitioners’ Agile-Methodology Use and Job Perceptions,” IEEE Software, vol. 35, no. 2, pp. 52-61, 2018.
14. K. Beck, M. Beedle, A. V. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, “Agile Manifesto,” 13 February 2001. [Online]. Available: http://agilemanifesto.org/history.html. [Accessed 27 February 2019].
15. M. V. Mäntylä, J. Itkonen and J. Iivonen, “Who tested my software? Testing as an organizationally cross-cutting activit,” Software Quality Journal, vol. 20, no. 1, pp. 145-172, 2012.
16. O. D. Otaduy, “User acceptance testing for Agile-developed web-based applications: Empowering customers through wikis and mind maps,” The Journal of Systems and Software, vol. 133, no. 1, pp. 212-229, 2017.
17. M. V. Mantyla and J. Itkonen, “More testers – The effect of crowd size and time restriction in software testing,” Information and Software Technology, vol. 55, no. 1, pp. 986-1003, 2013.
18. K. Beck, Test-driven development by example, Beaverton, Oregon: Ringgold Inc, 2003.
19. “DevOps is Simply Interaction Between Development and Operations,” in Devops 2018, 2019.
20. A. D. Nagarajan, “DevOps implementation framework for Agile-based large financial organizations,” Utrecht University, Utrecht, 2018.
21. J. Angara, S. Gutta and S. Prasad, “DevOps with Continuous Testing Architecture and Its Metrics Model,” Recent Findings in Intelligent Computing Techniques, vol. 709, pp. 271-281, 2018.
22. L. F. Albuquerque and F. S. Ferraz, “Function as a Service X Platform as a Service: Towards a Comparative Study on FaaS and PaaS,” in Intl Conference on Software Engineering Advances, 2017.
23. “Correlation of critical success factors with success of software projects: an empirical investigation,” Software Quality Journal, pp. 1-65, 2018.
24. X. Chen, D. Zhang, Y. Zhao, Z. Cui and C. Ni, “Software defect number prediction: Unsupervised vs supervised methods,” Information and Software Technology, vol. 106, no. 1, pp. 161-181, 2018.
25. R. V. Megen and D. B. Meyerhoff, “Costs and benefits of early defect detection: experiences from developing client server and host applications,” Software Quality Journal, vol. 4, no. 4, pp. 247-256, 1995.
26. G. Samarthyam, M. Muralidharan and R. K. Anna, “Understanding Test Debt,” in Trends in Software Testing, Singapore, Spring Nature, 2017, pp. 1-19.
27. C. Stringfellow, A. Andrews, C. Wohlin and H. Petersson, “Estimating the number of components with defects post-release that showed no defects in testing,” Software Testing Verification and Reliability, vol. 12, no. 1, pp. 93-122, 2002.
28. L. Mariani, D. Hao, R. Subramanyan and H. Zhu, “The central role of test automation in softwarequality assurance,” Software Quality Journal, vol. 25, no. 3, pp. 797-802, 2017.

29. W. R. Elmendorf, “Evaluation of the Functional Testing of Control Programs,” IBM, Pughkeepsie, New York, 1967.
30. P. Naur and B. Randell, “Software Engineering,” in NATO Science Committee, Garmisch, Germany, 1968.
31. P. Reales, M. Polo, J. L. Fernández-Alemán, A. Toval and M. Piattini, “Mutation Testing,” IEEE Software, vol. 31, no. 3, pp. 30-35, 2014 .
32. J. D. McGonagle, “Notes on the computer program test methods symposium,” ACM SIGPLAN Notices, vol. 7, no. 5, pp. 8-12, 1972.
33. D. S. Alberts, “The economics of software quality assurance,” in National Computer Conference, McLean, Virginia, 1976.
34. D. C. Bossen and S. J. Hong, “Cause-Effect Analysis for Multiple Fault Detection in Combinational Networks,” IEEE Transactions on Computers, vol. 20, no. 11, pp. 1252-1257, 1971.
35. W. P. Stevens, G. J. Myers and L. L. Constantine, “Structured Design,” IBM Systems Journal, vol. 13, no. 2, pp. 115-139, 1974.
36. N. S. Waldstein, “The Walk-Thru- A Method of Specification, Design and Review,” IBM, New York, 1974.
37. M. B. Zanjani and H. Kagdi, “Automaticall Recommending Peer Reviewers in Modern Code Review,” IEEE Transactions on Software Engineering, vol. 42, no. 6, pp. 530-543, 2016.
38. P. Thongtanunam and C. Tantithamthavorn, “Who should review my code,” in International Conference on Software Analysis, Evolution, and Reengineering,, Montreal, Canada, 2015.
39. C. Kaner, H. Q. Nguyen and J. Falk, Testing Computer Software, Wiley, 1988.
40. F. Asplund, “Exploratory testing: Do contextual factors influence software fault identification?,” Information and Software Technology, vol. 107, no. 1, pp. 101-111, 2019.
41. D. Geleprin and B. Hetzel, “The growth of software testing,” Communications of the ACM, vol. 31, no. 6, pp. 687-695, 1988.
42. Clarke, P., O'Connor, R.V., Leavy, B.: A Complexity Theory viewpoint on the Software Development Process and Situational Context. In: proceedings of the International Conference on Software and Systems Process (ICSSP), pp. 86-90, (2016)
43. Clarke, P., O'Connor, R.V., Leavy, B., Yilmaz, M.: Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance. IEEE Transactions on Software Engineering, 41(12), pp.1169-1183, doi: 10.1109/TSE.2015.2467388 (2015)
44. Yilmaz, M., O'Connor, R., Clarke, P.: A Systematic Approach to the Comparison of Roles in the Software Development Processes. In: proceedings of the 19th European Conference on Systems, Software and Services Process Improvement (EuroSPI 2012). pp.1-12.
45. Sauberer, G., Najera Villar, B., Dreszler, J.R., Schmitz, K.D., Clarke, P., O'Connor, R.V.: Do we speak the same language? Terminology strategies for engineering environments based on the elcat model. In: 24th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2017), pp. 653-666.
46. Clarke, P., Mesquida Calafat, A.L., Ekert, D., Ekstrom, J.J., Gornostaja, T., Jovanovic, M., Johansen, J., Mas, A., Messnarz, R., Najera Villar, B., O'Connor, A., O'Connor, R.V., Reiner, M., Sauberer, G., Schmitz, K.D., Yilmaz, M.: Refactoring Software Development Process Terminology through the use of Ontology. In: Proceedings of the 23rd European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2016).
47. Clarke, P., O'Connor, R.V., Solan, D., Elger, P., Yilmaz, M., Ennis, A., Gerrity, M., McGrath, S., Treanor, R.: Exploring Software Process Variation Arising from Differences in Situational Context. In: Proceedings of the 24th European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2017), pp.29-42.

48. Clarke, P., O'Connor, R.V.: Changing situational contexts present a constant challenge to software developers. In: Proceedings of the 22nd European and Asian Conference on Systems, Software and Services Process Improvement (EuroSPI 2015), pp. 100-111, (2015)