Anais da 1º Escola Regional Engenharia de Software (ERES 2017)
em Alegrete, RS, Brasil, 20 de outubro de 2017

On the Characterization of the SDET Role in Agile Projects: Expected Skills, Responsibilities, and Practices

Carolina Toscani, Júlia Couto, Josiane Kroll,
Alessandra Dutra, Sabrina Marczak, Rafael Prikladnicki

[1] MunDDos Research Group - PUCRS
Porto Alegre, RS - Brazil

{carolina.toscani, julia.couto, josiane.kroll}@acad.pucrs.br

{alessandra.dutra,sabrina.marczak,rafaelp}@pucrs.br

Abstract. *Software Development Engineer in Test (SDET) is a formal title for a software tester in software development context. In this paper, we report on a systematic literature review on the SDET role in agile projects. From the 27 selected papers, we extracted SDET role' skills, practices and responsibilities. For instance, we found that motivation, flexibility, and work experience in software engineering are important skills for the SDET be able to perform her work. We also found that developing the system, performing automated unit test, and performing exploratory tests are among the main responsibilities of a SDET. Test automation, A/B test, and Behavior-Driven Design are some of the most cited practices of this role in agile projects.*

1. Introduction

The SDET - Software Development Engineer in Test, is a term originated in 2005 at Microsoft. Developers at Microsoft have the common title of Software Development Engineer, or SDE [Page and Johnston 2008]. Analogously, the develop features by writing code. Analogously, the formal title for a software tester at Microsoft is Software Development Engineer in Test, or SDET. The similarity in the names of the two disciplines is by design, because testers at Microsoft are developers.

As initially proposed by Microsoft and later adopted by other companies, a SDET is part of the development team, and it participates in the full development cycle process. This role must be able to create high quality, maintainable, and performance code. Among other things, testers design tests, influence product design, conduct root cause analysis, participate in code reviews, and write automation tests. Occasionally they check bug fixes or work on small features. Sometimes testers check bug fixes or work on small features. Testing is a heavy workload, so writing features is not very common for testers. In this context, there are many open questions about the SDET concept and its implementation in agile projects: What are the skills necessary to perform the SDET role? What are the responsibilities of the SDET role? What are the practices associated with the SDET role? To answer these questions we investigated the SDET concept in the literature. We searched for theoretical and empirical work, given the novelty of the topic.

Our results yielded 14 skills for a SDET person to perform the role. These skills are organized into two categories, namely: personal and technical. We also found 16 key responsibilities and 34 practices associated with the SDET role. The reminder of this paper presents our literature review and details our results.

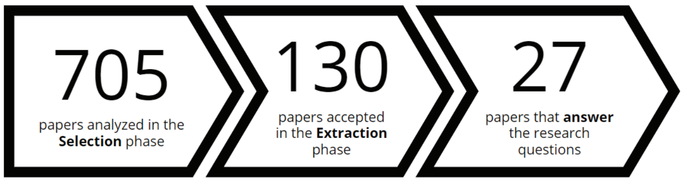
705
papers analyzed in the **Selection** phase
130
papers accepted in the **Extraction** phase
27
papers that **answer** the research questions

Figure 1. Phases of our literature review

2. Research Method

Based on the perceived need for conducting this study, our research questions (RQ) are as follows: What are the responsibilities of the SDET role? What are the skills necessary to perform the SDET role? What are the practices associated with SDET role? To answer the posed research questions, we searched studies using the Scopus digital library. We adopted the Start tool [Start 2017] for data extraction and to categorize the necessary information about study characteristics and findings from the included studies. The search was conducted using the boolean search expression as follows:

(ALL(”software development”) OR ALL(”software project”) OR ALL(”software engineering”) OR ALL(”software”) OR ALL(”system development”) OR ALL(”information system”)) AND TITLE-ABS-KEY(agile) AND ALL(automat*) AND (ALL(testing) OR ALL(test))) AND (LIMIT-TO(SUBJAREA,”COMP”)) AND (LIMIT-TO(LANGUAGE,”English”))

Figure 1 presents the phases of our literature review. After this extensive data search, 705 papers were identified, published between the years 1995 and 2016 - year in which the research was made. Upon a first round of reviews considering the paper title, abstract, and keywords, a total of 130 papers were selected for full reading. Out of these 130 papers, 27 were selected after the application of inclusion and exclusion criteria (second round of review). The inclusion and exclusion criteria for the study are as follows:

- (I) Papers of qualitative or quantitative nature of the subject on SDET.
- (I) Papers available online for download at no charge.
- (I) Papers with the full text of the study in electronic format.
- (I) Conference, workshop, and journals studies.
- (I) Empirical, theoretical papers.
- (E) Extended abstracts and editorial papers.
- (E) Papers unavailable online for download at no charge.
- (E) Papers not related to SDET.
- (E) Duplicated papers.
- (E) Non-English written papers.

3. Results

Our literature review identified 14 skills, 16 responsibilities, and 34 practices related to the SDET role in agile projects. To better relate these results to agile development, this section first describes the context in which the reported projects took place and next introduce each of the results per category: skills, responsibilities, and practices.

Context is the set of circumstances or facts that surround the software development where the SDET role acts. Most studies did not specify the companies' name in which this role was observed (74.1%), but Microsoft, Ericsson, and John Deere are cited. Most of the companies were in the domain of software development (33,3%), 7% are university studies, and 33,3% are undefined. Companies are located in Germany, Norway, India, Italy, China, South Korea, Brazil, and Finland.

The analysis show that in 75% of the papers developers are responsible for the tests. About 94% of selected papers (25 out of 27) presented testers as part of the team, while a minority presented testers working outside the team. About the adopted agile methods, 40,7% of papers cited Scrum, 25,9% cited XP, and 3,8% cited both Scrum and XP. It also shows that 44,4% of the papers described projects developed in a collocated environment, while only 14,8% are executed using distributed software development. More than 40% do not have information about how the teams interact physically, if they are or not in the same location.

Most of the projects (70%) do not inform how many teams are working in the projects, and less than 30% have between 1 and 3 teams working together. 75% papers do not give information about the number of people in the same team, neither. In the other ones, the number varies from 3 to 19 people working together in the same team.

3.1. SDET Skills

A total of 14 skills were identified. SDET skills can be organized into two groups of characteristics: personal and technical. Personal skills are the individual characteristics of a person, mostly related to attitude and personality. Technical skills are abilities acquired during training or experience, such as a specific programming language, test techniques, a technique used to get systems requirement, etc. It is important to mention that a SDET may have all personal and technical skill or at least some of them.

According to the literature, people who are SDETs must be *highly motivated*. The responsibility sense and the positive feedback can contribute to high levels of motivation. Motivated people think about their tasks as challenges, and do not care if they have to redo the tasks some time, to make it better. They think about new ideas, so the time they spend at work is not waste just in routine stuff. On the other hand, when someone is not motivated to do some task, the tasks can be annoying, and upsetting [Prechelt et al. 2016].

Other personal skill related to SDET role is *responsibility*. It is one of the consequences of empowerment. Every team is responsible for what they release. It is most related to the responsibility that a developer have to have on the code he develops, so one cannot write a code knowing that it will fail and rely that other person (the tester) will find and fix it [Prechelt et al. 2016]. Additionally, SDETs also should be *determined, flexible,* and *talented*.

As Agile teams usually work with the premise that everyone on the team have a broad skill set, the following technical skill are emphasized:

- Knowledge of code development: know how to use the programming language.
- Knowledge of database administration: knowledge on managing, installing, update and monitoring a database.
- An understanding of system analysis: process mapping, data modeling, and requirements gathering in a system development project.

- Knowledge of database architecture: data modelling in a database.
- Ability to work with interface design: design of the system interaction and user experience.
- Work experience in software engineering: processes related to software specification, development, and maintenance.
- Knowledge of project management: application of knowledge, skills and techniques to the execution of projects in an effective and effective way.
- Knowledge of systems engineering: development and organization of complex artificial systems.

3.2. SDET Responsibilities

Responsibilities are tasks or set of tasks attributed to the SDET role, that are executed individually or in collaboration with other roles. Some responsibilities such as performing automated unit test, developing the system [Agarwal and Deep 2014], performing integration tests [Sultanía 2015], and writing test cases [Little et al. 2012] are only cited in the papers. Others are described in details as presented next.

- Refactoring code: this responsibility is based on rewriting the code to match with acceptable standards [Kim and Ryoo 2012].
- Performing automated test: it is made in order to perform various testing activities, such as: test case management, test monitor and control, test data generation, test case generation, and test case execution [Rahman et al. 2015]. It can be used as a real-time indicator of the development progress and as a live documentation on the software [Park and Maurer 2008].
- Performing acceptance testing: it is an agile testing approach, also known as automated acceptance testing (AAT). In this approach, the customer or her representative is given the role of expressing requirements as input to the software team paired, with an acceptance test as a result. Acceptance testing allows to test business logic. It is a way of communicating precise requirements from the customer to the developers' point of view [Haugset and Hanssen 2008]. Acceptance tests are written in early stages, in order to validate user needs and it helps to direct the codification [Abrahamsson et al. 2009].
- Performing exploratory testing: it is a type of manual test, in which the responsible for the test utilize their domain expertise to explore and challenge the system in a way more likely to be used by one of the real users [Little et al. 2012].
- Performing fixtures test: it is strictly related to the software platform, and it can be used as formal specifications. The developer writes test to legacy systems, then migrates it to the new system, modifying or rewriting the code according to the target platform. The fixture tests should be written to test both: the legacy and the target systems [Abbattista et al. 2009].
- Writing positive and negative tests: negative tests are tests written to provoke an error, and it is used mainly to ensure that the system handles errors as intended. Positive tests are the common type of tests [Haugset and Hanssen 2008].
- Participating in requirements definition: the developers substantially participated in the requirements definition, apparently based on (1) their increased understanding of the domain as produced by more realistic feedback from the field (as discussed below), (2) their generally high motivation (also discussed below), and (3)

their feeling of responsibility. Their participation reinforced the motivation and the feeling of responsibility [Prechelt et al. 2016].
- Performing tables test: once story test tables have been defined, developers write test fixtures – the code which is used to take the inputs from the story test tables – exercise the system under test (SUT) and compare the observed and expected results [Abbattista et al. 2009].
- Writing test specification: developers will start from scenarios produced in one iterative decomposition process. Scenarios will be translated to tests, which will drive the implementation. A scenario is composed of several steps. A step is an abstraction that represents one of the elements in a scenario, which are: contexts, events, and actions. [Solis and Wang 2011].
- Performing unit test: once user stories are ready, customers or users together with testers or developers define story tests aimed at validating requirements described by user stories. Having the tests in place gives a clearer perspective on what one wants to achieve as well as a confidence in performing the migration. Differently to test fixtures, which are strictly related to the software platform, story tests are written just once and they should be completely or, at least, partially reused on the target system after they are written on the legacy system [Abbattista et al. 2009].
- Writing user stories: it aims building a descriptive form of the requirements of the system to be migrated. User stories might also consider issues related to the specific platform for which the migration is executed [Abbattista et al. 2009].
- Performing white-box test: in Agile methodologies, developers are in charge of executing white-box test using unit testing as a part of the validation process. As developers are working with the source code, they know: the software structure to define the appropriate test cases and, when a test case fail which lines of code are are involved in the revealed a problem [Abrahamsson et al. 2009].

3.3. SDET Practices

In the software engineering area, a practice is a software development practice that, through experience and research, has proven to reliably lead to a desired result and is considered prudent and advisable to do in a variety of contexts. We found 34 practices related to SDET role, presented in detailed next. However, three practices were only cited in the papers, with no associated description, as follows: exploratory testing [Anand and Mani 2015], refactoring [Tripp et al. 2016], and user stories [Almog and Tsubery 2015].

- Automated testing: it is the practice of automated techniques in order to perform various testing activities, such as test case management, test monitor and control, test data generation, test case generation, and test case execution [Rahman et al. 2015]. Test automation is a software framework that helps to test software product. It allows the team to accumulate test cases over the life of the application, so that both existing and new features can always be tested [Sultanía 2015].Verify definition in the papers [Sultanía 2015, Rahman et al. 2015].
- Collaboration with stakeholders: it is about practices of collaborations between groups of stakeholders, such as product managers, senior managers, user manual authors, application specialists. One example is participation in trade shows, where the test engineers have the opportunity to interact with customers and vendors [Anand and Mani 2015].

- A/B testing: also known as Alpha Beta test, it is a type of test in which a specific version of the software is deployed to a limited number of users to get feedback [Rahman et al. 2015].
- Automated Acceptance Testing (AAT): is used to test if the software developed comply with the contractual requirements of the end user [Rahman et al. 2015]. It relies on automation, and provide specification details[Kim and Ryoo 2012]. It is a complementary test approach in agile development.
- Agile specification driven development: it is an extension of TDD (Test-Driven Development), combined with Design-by-Contract (DbC) that allows a language to explicitly declare pre and post-conditions for invariant and methods for a class [Hammond and Umphress 2012].
- Reproduce concepts and theories: it is concerned to applying learnings from conferences, trade shows, courses, etc. in the tester engineer daily routine [Anand and Mani 2015].
- Automated builds: to rebuild the software, the team use a code script. Using this practice, they can assure that all the files that are necessary to successfully build the software are being added to the code repository [Tripp et al. 2016].
- Behavior-Driven Design (BDD): aims to change the mindset of the developers to focus on the behavior of the code [Hammond and Umphress 2012]. Its objective is to get executable specifications of a system [Solis and Wang 2011].
- Bring-up tests: test cases made to verify the most basic functionality of the application. The build is tested just after the bring-up test run successfully [Anand and Mani 2015].
- Competency development: as test engineers has to become generalizing specialists, competency development plans has to be defined. For example, a training needs list is defined, as well as a list of people who have the competency to provide training, and it results in training schedules. It can also be auto developed, by listening/watching audio and video records [Anand and Mani 2015].
- Continuous integration: using this process, the code is systematically and regularly built and deployed to a test server [Tripp et al. 2016, Williams et al. 2011]. In continuous integration practice, the work created by the members of a team is frequently integrated, normally each person integrates at least daily, and it leads to multiple integration per day [Kim and Ryoo 2012].
- Customer-centric configuration information: are identical replicas of the client systems configuration and it is formed by identical hardware in the customer site, operating system, third-party software, patches, and appropriate database [Anand and Mani 2015].
- Adoption of EATDD or ATDD or STDD: in Executable Acceptance Test Driven Development (EATDD), software requirements are written in form of executable acceptance tests, which are developed by the customer in collaboration with business analysts and/or quality assurance engineers [Park and Maurer 2008]. Acceptance Test-Driven Development (ATDD) is one approach in which the customer writes acceptance tests that the developers can use to see if their software is providing the correct functionality [Hammond and Umphress 2012] [Smith et al. 2009].
- Exposure to testers: to expose the test engineer to the end user work-flow, training can be provided in application workflows, customer site visits may be arranged, and weekly meeting with customer care personnel can be scheduled. Others forms

of exposures are related to awareness of competitor's solution and establish or improve communication with Product Management [Anand and Mani 2015].

- Focus on high impact errors: every test engineer attempt to create the highest impact, by testing only as much as necessary, in order to make testing effective. The code changes that have the highest impact on the entire functionality are prioritized [Anand and Mani 2015].
- Functional test: in this type of test, functional requirements of the software are tested with a specific set of input, ignoring the internals of the software [Rahman et al. 2015].
- Global test planning: it is derived from the test concept, that specifies the test approach, the major test assets required and their availability and the strategy for adhere to regulatory requirements and acceptance with customers. The definition of the test concept requires a deep understanding of the domain, technology expertise, and awareness of the regulatory requirements [Anand and Mani 2015].
- Increasing quality focus: to reduce unproductive effort spent on testing, adherence to quality, regulatory and organization quality standards is a very important step way to achieve this goal is providing testers with the same access to team activities, documentation and portals, just like the others team members [Anand and Mani 2015].
- Integration test: refers to gradual testing among different components of the software to test the functionality of the complete software [Rahman et al. 2015].
- Involving the customers: keeping they well informed. For example, when done in moderation, it was possible for one team in Europe to interact and align with the customers due to their proximity, on a regular basis seeking feedback through product demo and informal verification of fixes at customer sites. Controlled customer involvement is important an agile contribution, but attention must be paid to all stakeholders [Anand and Mani 2015].
- Training plan in applications: it typically takes a long time to learn the complex domain of the applications. The complexity is in the terminology, conceptual understanding and a wide range of interfacing systems. A training plan involving all the basics was imparted through instructor-led training's, supplemented with online training material [Anand and Mani 2015].
- Pair programming: this practice recommends putting two developers working in pairs to develop a portion of code [Tripp et al. 2016].
- Planning meetings: involves to plan a meeting with the entire global team for test team coordination and stand-up meetings covering different locations [Anand and Mani 2015].
- Regression suite/test: includes functional and non-functional tests. Some regression test are automated. However, several regression tests are executed manually, such as simulating customer-centric configurations and testing for interoperability, using the approach of "warm body testing'. To ensure the efficiency, it was scheduled only when the application was sufficiently stable [Anand and Mani 2015]. Regression testing is performed to check if a software change has adversely affected the functionality of the whole software [Smith et al. 2009].
- Repository Driven Test Automation (RDTA): proposing reuse of testing artifacts as a fundamental principle for the creation test automation is what makes RDTA a unique approach to the buildup process of test automation infrastructure. The

term software repository here refers to a storage location from which software packages or artifacts may be retrieved to be reused on another system or other software products [Almog and Tsubery 2015].

- Achieving Sprint/iteration level planning: some user stories were large and spanned multiple sprints. Scrum teams broke these down to smaller user stories and defined the done criteria for these smaller user stories. This enabled assessing the degree of completion of planned tasks for the sprint and in coordinating activities [Anand and Mani 2015].
- Test Driven Development (TDD): involves the implementation of a system starting from the unit test cases of an object. After writing automated test cases that generally will not even compile, the programmers write implementation code to pass these test cases. The work is kept within programmer's intellectual control; as the programmer is continuously making small implementation decisions and increasing functionality at a relatively consistent rate. All of the test cases that exist for the entire program must successfully pass before new code is considered fully implemented [George and Williams 2004].
- Testing only on Sprints: for the 'test-only' sprints, the test case selection is based on availability of fixes, impacted features, and new test cases created since the last testing cycle, new environment, and test data. In these sprints, complete regression testing at system level is done, as derived from the guidance from the regulatory agencies. In this phase all members of team are involved in testing [Anand and Mani 2015].
- Testing early: before moving to agile, the number of integration tests was lower than the number of system tests. With the aim or executing more test cases early in the life cycle, they tried to identify tests that could equivalently test the feature at an earlier phase. If it was possible for the objective of a system test case to be realized at the integration level, the system test case was moved to the integration test phase [Anand and Mani 2015].
- Training testers on test case design: Senior Test Team members trained for Test Engineers on test case design. This practice suggest the following activities:
  - Use of flowcharts and tables was encouraged during test case design.
  - A set of "do's don'ts" have helped in a uniform understanding of the test case design methodology.
  - Test objective was stated precisely matching the requirements being tested and not as generic statements, which don't imply anything specific.
  - Risk mitigation test cases were treated as a separate set of test cases to give better focus.
  - The data-flow test cases identified and recorded all application attributes and their combinations that would be passed through, on test execution.
- Training related regulatory requirements: a rigorous enforcement of requirement tracing and collecting evidence of testing through screen shots of results and logs during test execution, clear document versioning, keeping a record of document reviews help in complying with regulatory requirements [Anand and Mani 2015].

4. Final Remarks

The Software Development Engineer in Test (SDET) is a formal title for a software tester in the context of software development. In this paper, we conducted a Literature Review for the period between 1995 and 2016. As a result from the search, we select 130 papers for further analysis and 27 papers that selected. Our results report the expected skills, responsibilities and practices related to SDET role.

SDET is highly skilled resource with development and testing skills. However a tester is involved in preparing and executing the test cases either manually or automatically. Testers are resources with limited programming skills and are focused on functional tests, while SDETs are skilled resources with good programming skills and do the job of tester as well as a developer in the automation of test. Testers are not expected to develop test automation tools, they can use this test automation tool to automate the test cases required for your software application or project. A SDET is proficient in software development, and them can participate in the development of test automation tools and can make it for generic use.

We learned that some of the Expected personal skills of a SDET are highly motivated and responsibility and the technical skills are knowledge of database administration and understanding of system analysis between 14 skills. There are 34 practices associated with the SDT role, but the practice is depend on the context of project is in, but Automated Test and collaboration with stakeholders showed as the most discussed.

It is important to highlight that our results are limited in the sense that are searched only for papers available in databases we have access to. It might be we left on to behind.

5. Acknowledgments

This work is part of a Research Project, sponsored by the PDTI Program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91) under the Dell/PUCRS Research Agreement.

References

Abbattista, F., Bianchi, A., and Lanubile, F. (2009). A storytest-driven approach to the migration of legacy systems. In *Proceedings of the International Conference on Agile Software Development*, pages 149–154, Pula, Italy. Springer.

Abrahamsson, P., Marchesi, M., and Maurer, F. (2009). *Agile Processes in Software Engineering and Extreme Programming*. Springer, Pula, Italy.

Agarwal, N. and Deep, P. (2014). Obtaining better software product by using test first programming technique. In *Proceedings of International Conference-Confluence The Next Generation Information Technology Summit*, pages 742–747, Noida, India. IEEE.

Almog, D. and Tsubery, Y. (2015). How the repository driven test automation (rdta) will make test automation more efficient, easier & maintainable. In *Proceedings of India Software Engineering Conference*, pages 196–197, Bangalore, India. ACM.

Anand, T. and Mani, V. (2015). Practices to make agile test teams effective: challenges and solutions. In *Proceedings of International Conference on Global Software Engineering Workshops*, pages 7–11, Ciudad Real, Spain. IEEE.

George, B. and Williams, L. (2004). A structured experiment of test-driven development. *Information and software Technology*, 46(5):337–342.

Hammond, S. and Umphress, D. (2012). Test driven development: the state of the practice. In *Proceedings of the Annual Southeast Regional Conference*, pages 158–163, Tuscaloosa, USA. ACM.

Haugset, B. and Hanssen, G. K. (2008). Automated acceptance testing: A literature review and an industrial case study. In *Agile Conference*, pages 27–38, Toronto, Canada. IEEE.

Kim, E. and Ryoo, S. (2012). Agile adoption story from Nhn. In *Proceedings of the Annual Computer Software and Applications Conference*, pages 476–481, Izmir, Turkey. IEEE.

Little, T., Elliott, S., Hughes, J., and Simion, F. (2012). Leveraging global talent for effective test agility. In *Agile Conference*, pages 165–171, Dallas, USA. IEEE.

Page, A. and Johnston, K. (2008). *How We Test Software at Microsoft*. Microsoft Press.

Park, S. S. and Maurer, F. (2008). The benefits and challenges of executable acceptance testing. In *Proceedings of the International Workshop on Scrutinizing Agile Practices or shoot-out at the Agile corral*, pages 19–22, Leipzig, Germany. ACM.

Prechelt, L., Schmeisky, H., and Zieris, F. (2016). Quality experience: a grounded theory of successful agile projects without dedicated testers. In *Proceedings of International Conference on Software Engineering*, pages 1017–1027, Austin, USA. ACM.

Rahman, A. A. U., Helms, E., Williams, L., and Parnin, C. (2015). Synthesizing continuous deployment practices used in software development. In *Agile Conference*, pages 1–10, Washington, USA. IEEE.

Smith, M., Miller, J., and Daeninck, S. (2009). A test-oriented embedded system production methodology. *Journal of Signal Processing Systems*, 56(1):69–89.

Solis, C. and Wang, X. (2011). A study of the characteristics of behaviour driven development. In *Proceedings of Conference on Software Engineering and Advanced Applications*, pages 383–387, Oulu, Finland. Austrian Computer Society.

Start (2017). Start — lapes - laboratório de pesquisa em engenharia de software. "`http://lapes.dc.ufscar.br/tools/start_tool`" -Accessed in September 2017.

Sultanía, A. K. (2015). Developing software product and test automation software using agile methodology. In *Proceedings of the International Conference on Computer, Communication, Control and Information Technology*, pages 1–4, Hooghly, India. IEEE.

Tripp, J. F., Riemenschneider, C., and Thatcher, J. B. (2016). Job satisfaction in agile development teams: Agile development as work redesign. *Journal of the Association for Information Systems*, 17(4):267.

Williams, L., Brown, G., Meltzer, A., and Nagappan, N. (2011). Scrum + engineering practices: Experiences of three microsoft teams. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 463–471, Washington, USA. IEEE.