International Conference on Emerging Trends in Engineering, Technology, Science and Management
IIMT College of Engineering, Greater Noida, India (ICETETSM-17)
12th April 2017, www.conferenceworld.in ISBN: 978-93-86171-38-2

SOFTWARE PRODUCTION ISSUES: ROLE AND SETBACKS OF QUALITY ASSURANCE TEAM IN AN AGILE ENVIRONMENT

Aishwarya Vatsa[1], Shiv Kumar[2]

[1,2]ICFAI Technical School, The ICFAI University, (India)

ABSTRACT

QA role has undergone major changes from traditional SDLC to agile SDLC. However, the prime expectation from them (QA) has remained unaffected, which is to provide defect free software to the clients. The path that any software follows, starts from developer environment, goes into QA environment and ends in the production environment. Here, issues which arise in the production environment are termed as Software Production Issues. To overcome these issues QA perspective needs exploration. Consequently, this paper presents the role of QA in agile environment and also discusses associated setbacks. The realisation of QA setbacks will further help in reduction of Software Production Issues.

Keyword: - Agile SDLC, QA role, Software Production Issue.

I. INTRODUCTION

Agile Environment is a collection of frameworks which promises to deliver Software Work Products (SWP) frequently and in regular time intervals (known as Sprint) [1]. The delivery process follows a full cycle consisting of SWP development, testing, bug fixes and verification. The Quality Assurance team (QA) is responsible for testing and verifying the SWP. In this paper, the role and responsibilities of QA are analysed to uncover the setbacks through which SPI (Software Production Issue) is originated. QA plays an important role in agile Software Development Life Cycle (SDLC). They are supposed to complete the verification process of the SWP, along with maintaining its quality. However, occurrence of issue in deployment leads to failure and bring QA in a questionable situation [2][3]. All the issues occurring at the time of SWP deployment are termed as SPI, which are likely to occur in the production environment (where environment's configurations are different).

In general, software testing is the prominent technique used for issue identification and removal. QA proposes a set of tests to be conducted over the provided SWP. The tests are executed and if any defect is discovered then, it is further provided to the developers for removal.

The process undertaken by QA is of verification and certification of issue. Though, all the efforts made by agile team members are ruined if an issue occurs at the time of deployment or in the production environment. Hence, to identify SPI at their origin and to handle them at their occurrence, QA's viewpoint is required. The major

problem with SPI is its criticality, as it occurs in the user/client environment. It is vital to understand that SPI could lead a software project to its closure. Issues occurring at any of the intermediate phase of SDLC is realised and removed. Their occurrence at the concluding phase of SWP development leaves every personnel related to the container project in dilemma [4][5][6][7].

The two characteristics of SPI; severity and priority, is the deciding factor for its treatment. An issue must be kept at high priority if its severity is high as well. Consequently, SPI severity and priority needs early realisation. SPI could impact any of the functionality of the SWP. It could also be a cosmetic issue which does not affect SWP's functioning. But, who is to question if it occurs at the worst of its scenario. Any issue is to be realised by the QA team. Occurrence of issue in SWP when it is Out Of QA Cycle (OOQC) leaves them in doubt. Hence, QA perspective is required for SPI resolution. It is not that only QA is to blame for SPI. There exist some scenarios where developers and operation team is responsible for SPI as well. Nonetheless, QA plays significant role in SPI identification and resolution. In this paper, QA role is analysed in agile SDLC. Their responsibilities are uniformly arranged, which helps in recognising areas where they lack. The setbacks associated with the QA team lead to further identify the origins of SPI.

In this paper, section II explores the role of QA in agile SDLC. Obstructions associated with their role are discussed in section III. Section IV, presents the conclusive key points of this paper.

II. ROLE OF QA IN AGILE SDLC

QA role in agile SDLC begins when the changes/updates in the SWP is deployed in their respective environment [8]. It is then followed by various testing measures undertaken by QA over the SWP, verification procedure and certification of the conducted tests. Afterwards, SWP is out of the QA cycle and ready for deployment in the client's environment. Therefore the scope of issue identification and resolution lies at the very end of OOQC.

2.1 Deployment of changes in QA Environment

After the SWP is developed or updated (with feature enhancement) by the developers, it gets extended to the QA. The team of QA then deploys this release in their environment by following the deployment steps which are documented by the development team. Following are the steps undertaken in this phase:

a) *Deployment Steps:* Steps to be followed by QA team for deploying SWP is defined and documented by the development team.

b) *Deployment Testing:* If the above defined steps are incorrect in any manner, then it will be uncovered while deploying the SWP in the QA environment. If there are no concerns raised during the deployment, then it implies that the change will be deployed in the client's environment successfully.

c) *Sanity Testing:* After deployment completion, QA conducts a quick sanity test with few basic test cases to ensure that there is no impact on the original functionalities of the SWP. This test consists of verification of start-up, error logs, directory structure, etc.

d) *Environment Verification:* Many of the configurations of different environments (Of developer, of QA and of final production) are different as well. In such cases, it is the responsibility of QA to ensure correctness

of configuration before SWP deployment in any of the environment. If there is a mismatch in the configuration, then it will lead to unpredicted behaviour of the SWP.

After following the above stated steps, the SWP is deployed in QA's environment and is ready for further verification and certification.

2.2 Different testing techniques used by QA

As mentioned above, the QA team depends majorly over testing for verifying SWP. There exist various testing techniques with their own set of characteristics which are employable in accordance with the QA. Following is a brief description of prominent testing techniques in the context of this paper [9][10][11][12][13]:

a) *Functional Testing:* This includes the execution of the test cases which are in accordance with the functional changes made in the SWP in the current release. After the completion of these tests, all functional changes made are accepted.

b) *Regression Testing:* It is a process of execution of regression suite, which is a collective set of test cases covering the whole of the SWP. The test cases are constructed to ensure that new functionalities do not impact the existing functionalities of the SWP. It is required to be done before SWP deployment in its respective client's environment.

c) *Automation Testing:* It is the process of automated execution of test cases to save manual efforts. Generally, regression test suites are automated to save QA efforts.

d) *Performance Testing:* It certifies the quality of SWP in terms of performance numbers. After the completion of performance testing, the numbers compare with the previous version to compute the difference, which indicates an increase or decrease in the performance of SWP.

e) *Backward Compatibility Testing:* This test is conducted when SWP is dependent over multiple servers (in a SWP having a dependency over the network). Use of multiple servers helps in load diversion in case of failure. Deployment strategy in such scenarios is to deploy the SWP at a single site and then to the next. There may be a case in which all or few of the servers are pointing to the same database. In this case the backward compatibility test is conducted for verification of the new/old SWP with old/new database utility.

f) *Rollback Testing:* There exists no such way which ensure that deployed SWP will be defect free. In few of the situation rollback is required to the previous state of the SWP. In such cases, a rollback plan is made which is then followed by the QA team to ensure successful rollback. Rollback testing is required to be done in these cases.

g) *Resilience Testing:* This testing technique involves an assortment of rainy day cases [] which occurs when the SWP is in running state. Few of these cases are:

- Misconfigured Instances
- Database Failure
- Network Failure
- Failover cases

There are many other testing techniques which are performed over the SWP to be released, other than the above testing techniques. However, in general, they are sub categorised under the above explained testing techniques.

2.3 Verification and Testing

In agile SDLC, there is a strict time constraint for SWP development and testing. In conformance with this challenge, it is unfeasible to perform all the available testing techniques before deployment. Consequently, the collection of tests to be conducted over the SWP depends on the changes and updates made in that particular sprint [14]. Table I is the representation of changes made to an SWP, special cases and required verification. Following are the types of changes made to the SWP:

a) *Functional Change:* This case arises whenever there are updates/changes made to the SWP, impacting its functionalities.

b) *Version Enhancement of Utility Module:* Updates are made to these modules for improvement of SWP's performance. However, QA needs to assure that existing functionalities of SWP does not get affected and the performance numbers do not go below the agreed upon threshold.

c) *Code Refractor:* Refactoring is done at the code level, when all the code does not require smooth execution. After code refraction, QA needs to assure functionalities working and performance numbers of the SWP.

d) *Defect Fix:* After every defect fixture by the development team, the functionalities of SWP require a check. This is done to assure that defect fix has not affected the SWP.

e) *Configuration Changes:* These are the changes made to the configuration files of a particular SWP. It is however not supposed to impact the performance numbers and functionalities of the SWP.

f) *Hardware Upgrades:* To improve the present functionalities of SWP, hardware are updated with the latest compatible support. However, if any, changes made to the environment are not compatible with the SWP, and then it could result in failure. Hence, hardware updates require specific verification process.

2.4 Change Certification

At the end of all the verification procedure followed by QA (in accordance with the forthcoming deployment), the QA needs to certify the changes made to the SWP. Afterwards, QA signs off the testing document, which implies that the present release is bug free and ready for deployment (in the client's environment). Table II represents the elements present in the testing document. The testing document consists of various attributes which establishes the factual details about the SWP. This document is further helpful for QA and development team.

After all the efforts put in by the QA team, there still exist, chances of occurrence of issues. These issues become crucial when they occur during the deployment phase or in the production environment. Fig. 1 consists of the categorised form of SPI along with their subcategories [15][16]. To overcome these issues, the areas where QA lacks needs exploration. The next section discusses all the setbacks associated with the QA team.

III. SETBACKS ASSOCIATED WITH THE QA

After a certain release goes through complete testing life cycles, all the issues get reported, fixed and verified. If any issue prevail to exist and reaches the production environment, then it may impact the working of SWP. From Fig. 1 issue are defined as following:

ISBN: 978-93-86171-38-2

TABLE I

S. No.	Changes (Requirements) over SWP	Cases	Tests to be performed
1.	Functional Change	-Feature enhancement -New Feature Addition -Change in Application flow -API/GUI change -New UI -Logging/ Reporting changes	-Functional Testing -Automation Testing -Regression Testing -Performance testing -Deployment Verification -Rollback verification
2.	Version Enhancement of the Utility Module	-Database Upgrades and Changes -Server Upgrade -Tools Upgrade -Jar Upgrades -Platform Change	-Performance Testing -Regression Testing -Deployment Verification
3.	Code Refractor	-Unused Code Removal -Code Enhancement -Coding Formatting -Coding Standards	-Performance Testing -Regression Testing -Deployment Verification
4.	Defect Fix	-Functional Defect -Cosmetic Defect -Performance Defect -Configuration Defect -Deployment Defect -Loggers	-Deployment Verification -Functional Verification -Regression Verification -Performance Verification -Logger Verification -Environment Verification
5.	Configuration Change	-Changes in configuration properties -Addition of new Properties -Removal of Unused Properties -Changes in Environment Variables	-Functional Verification -Regression Testing -Performance Testing -Deployment Testing -Rollback Verification
6.	Hardware Upgrade	-Increase in RAM size -Upgrade to the latest OS Version -Increasing the Disk Space -Increase in the number of processors -Increase in the number of cores	-Sanity Testing -Regression Testing -Performance Testing -Deployment Verification -Rollback Verification

		-Cache support improved -Increase in heap memory -Server upgrades -Change in cloud platform	-Environment Verification

TABLE II

S. No.	Information	Explanation
1.	Functional Test Cases	Summary of test cases which ran against the release
2.	Features Covered	Features of the SWP, which were covered during testing
3.	Features not covered	The Unaffected features which were not tested
4.	Type of Testing Performed	All the types of testing performed over the SWP
5.	Type of Testing not Performed	These are the types of testing, which were skipped in the verification cycle
6.	Assumptions	For every verification process, there are some assumptions made
7.	Testing Environment	Environment details under which testing has been performed
8.	Testing Environment, v/s Production Environment	For future reference, the difference between testing and production environment is recorded
9.	Open Defects	Defects which were not fixed in the present sprint
10.	Release Version	There can be many releases for a single change with different version numbers.
11.	Performance Numbers	This consists of the performance details of the SWP
12.	Conclusion	The conclusion of whole of the testing procedure
13.	Recommended Configurations	It consists of the configuration details under which the release has been certified

a) *Functional Issue:* These are sunny day, rainy day or edge cases which could hit the production and impacts the functionalities of SWP. Some of these issues are easy to replicate and test, some of them are not.

b) *Legacy Issue:* These are the issues which get carried over from previous sprints to the present sprints. They are low priority issues as their presence in the SWP does not impact its functionalities. They could be few unidentified or low priority performance issues.

c) *Performance Issue:* These are the issues which directly impacts the performance of SWP in the live environment. They could be memory issue, load issue, CPU utilisation issue, connection leak issue or third-party module behaviour issue.

d) Environmental Issue: These are the issues which occur due to differences between the various associated environments of SWP. These environments could be of developer, QA or production. Whenever there is a mismatch between the variables associated with these environments, an issue is generated.

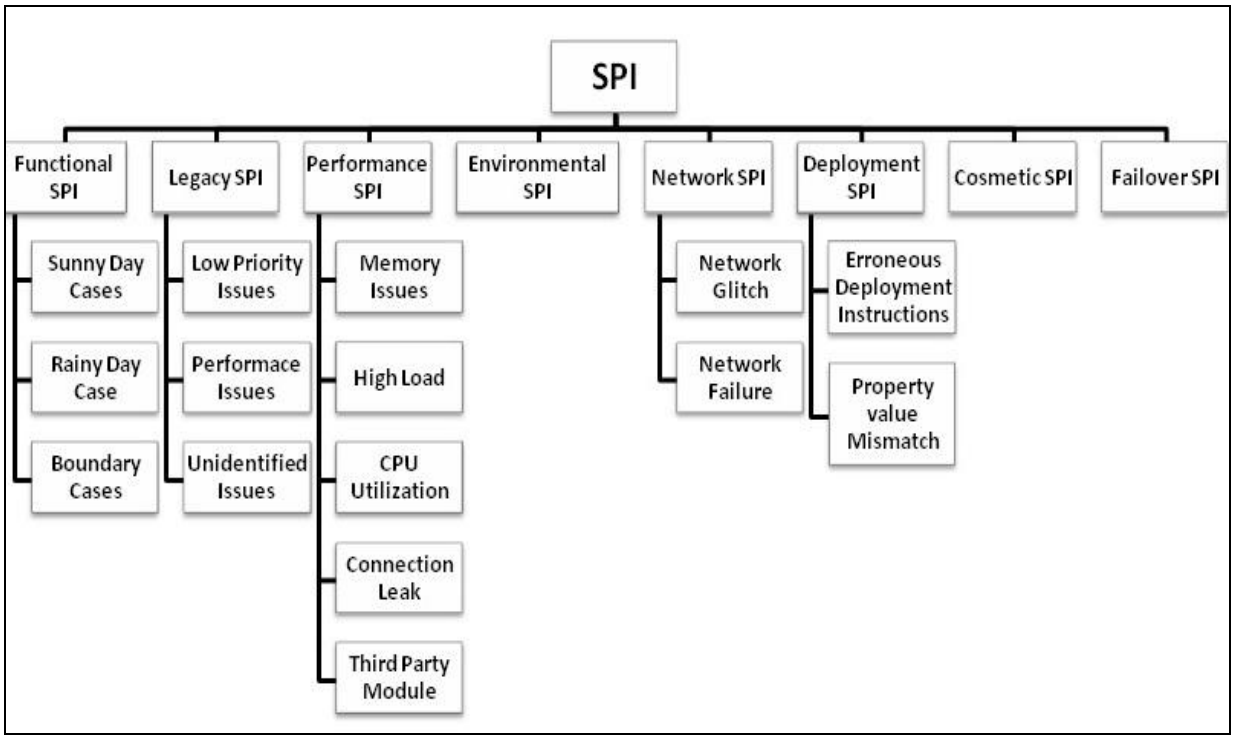

Figure 1 SPI Categories

e) Network Issue: Many web applications (SWP) have connections over the network. There continuous execution requires these connections to be maintained. Network glitches and failure may cause an issue in the production environment.

f) Deployment Issue: The already set steps involved in the deployment process, if not followed in sequence, may produce an issue in the production environment [17][18]. This issue is the result of erroneous deployment instruction or application properties' values mismatch.

g) Cosmetic Issue: It is an issue which directly affects the layout or GUI of the SWP. Generally, they are low priority issue, but can complicate equations with customers if the SWP has a dedicated GUI for customers' interactions.

h) Failover Issue: These issues which occur due to the dependencies between the primary and secondary modules of SWP and are handled at the product's code level.

The above issues are to be raised and resolved in the testing phase. However, due to some unknown factors these issues remain unnoticed and get a pass to the production environment. If there exist any way to solve them, then it is through the QA team. Hence, understanding that where the QA lack is vital for SPI resolution as they are the origin of SPI. The following are the ways in which QA lacks in uncovering issues beforehand:

a) Environmental Difference: There always exist minor or major differences between the environment of SWP development/ testing and the environment in which it is to be deployed. These differences alter the

variables' values and create chaos in the production environment. The parameters of any environment are: RAM, disk space, OS properties, number of cores, number of processors, latency, etc.

b) *Regression Execution:* In Agile SDLC, before every release regression automated suite is executed for confirming that the present changes done to SWP does not affect the existing functionalities. However, there might be cases which have not been covered by the regression suite due to edge issues or technicality involved. These cases may be impacted by the present changes and generate an error in the production environment. The reasons for non-inclusion of these cases could be following:

- Non-automatable Cases: There are few cases which cannot be automated due to technical constraints and hence are not included in the regression suite.
- Edge Cases: There are few cases which lie at the edge and have very probability of occurrence.
- Unidentified Cases: The QA team miss out few cases which were to be automated and tested.
- Time Taking Cases: There are cases which when automated takes too long for execution and so to keep an optimum time limit for regression execution, these cases are not included in the suite
- Dynamic Test Data: If the output of test cases is dynamic in nature, then they are not included in the regression suite. The non-inclusion is because there is the requirement of matching actual and expected outputs in regression execution. Due to dynamic nature of the outputs, the matching cannot be done.
- Complex Flow: Few of the cases have a complex flow and so are not in the regression suite.

c) *Functional Cases:* The identification of test cases for functional testing is the prior task before every release. Under this procedure, the actual output of test cases are evaluated against expected output. If the result of this evaluation is positive then it assumed that SWP is bug free. However, there are times when QA misses to identify these functional test cases. This will further result in patches of codes which remain unverified and the chances arise that it will create issues at the of production. Hence, to identify these cases, QA needs to be alert.

d) *Minimal Time Period:* The various frameworks in agile SDLC has generally a strict time duration under which the SWP needs to be developed and deployed. This time duration is termed as sprint. A sprint normally consists of 7 to 14 days. During this duration the development and testing teams is expected to develop the required features and also conduct tests to verify these. Under this strict time schedule, the quality of testing could be compromised. The QA team needs to be alert during the verification process of SWP.

e) *Third Party Module Failure:* A SWP may have dependencies over external modules. These modules are required not to fail during SWP execution as it may propagate and result in the failure of SWP itself. These scenarios, if left uncovered, then it may impact SWP in the production environment. However, the QA team is not solely responsible for this failure; but still; considerations must be made to handle these scenarios as well. Examples of third-party module failure are: database connection failure, cloud connection failure, etc.

f) *Network Glitch:* These are few of the cases which remain unverified in the production environment as they are difficult to replicate and test. Any scenarios which have dependencies over the network, comes under this case. So, they are generally treated as undefined exceptions.

g) *Lack of Domain Knowledge:* It cannot be expected from the QA to have all the knowledge about ever domain. Many of the time, it impacts the QA capability to test and verify SWP. Thus, QA require to have thorough domain knowledge of their respective work.

h) *Miscommunication:* Communication is the foundation of agile framework (as mentioned in agile manifesto) [19]. However, if there arise misunderstanding at the time of requirement gathering, then it could further lead to wastage of developers and QA efforts, and also unpredictable behaviour of SWP in production environment. The reason behind this is the mismatch between client's requirement and final SWP in the live environment.

The above are the areas which where QA lacks and due to which SPI is generated.

IV. CONCLUSION

In this paper, role of QA in agile SDLC has been discussed to its core. The objective of this study is to analyse the work of QA and their respective areas where they lack. These lacks generate the chances of issue (SPI) in SWP. However, before conceptualising the role of QA, SPI categorisation has been done. The essence of this paper is to study QA perspective towards developed SWP and how they offer to verify it. The reason behind this study was to identify the areas where SPI originate.

As, the SWP follows a path of Developer-QA-Production environment, so, the solution needs to be presented between QA to production environment. The study in this paper encourages this idea. Hence, an approach introduction is required to mitigate SPI and reform the role of QA.

REFERENCES

[1] Brown, A. W.; Ambler, S. And Royce, W., 2013, Agility at Scale: Economic Governance, Measured Improvement, and Disciplined Delivery, ICSE (IEEE): 873-881.

[2] Mäntylä, M. V. And Vanhanen, J., 2011, Software Deployment Activities and Challenges – A Case Study of Four Software Product Companies, 15th European Conference on Software Maintenance and Reengineering (IEEE): 131-139.

[3] Rathod, N. And Surve, A., 2015, Test Orchestration: A framework for Continuous Integration and Continuous Deployment, International Conference on Pervasive Computing (IEEE).

[4] Ruiz-Rube , I.; Dodero, J. M. And Colomo-Palacios, R., 2015, A framework for software process deployment and evaluation, Information and Software Technology (Elsevier): 205-221 (59).

[5] Schermann, G.; Cito, J; Leitner, P. And Gall, H. C., 2016, Towards Quality Gates in Continuous Delivery and Deployment, ICPC (IEEE): 1-4.

[6] Su, X.; Liu, H.; Wu, Z.; Zuo, D. And Yang, X., 2010, SA Based Software Deployment Reliability Estimation: Problem Space, Challenges and Strategies, International Conference on Educational and Information Technology (IEEE): 130-135 (2).

[7] Yim, K. S., 2014, Norming to Performing: Failure Analysis and Deployment Automation of Big Data Software Developed by Highly Iterative Models, IEEE 25th International Symposium on Software Reliability Engineering: 144-155.

[8] Kanij, T.; Merkel, R. And Grundy, J., 2014, An Empirical Study to Review and Revise Job Responsibilities of Software Testers, IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC): 89-92.

[9] Vatsa A., 2015, Experimental Study on Test Automation using Selenium WebDriver, JCT, Vol. 4, Issue 11.

[10] Collins, E.; Dias-Neto, A. And Lucena, V. F., 2012, Strategies for Agile Software Testing Automation: An Industrial Experience, IEEE 36th International Conference on Computer Software and Applications Workshops: 440-445.

[11] Hellmann, T. D.; Chokshi, A.; Abad, Z. S. H.; Pratte, S. And Maurer, F., 2013, Agile Testing: A Systematic Mapping across Three Conferences: Understanding Agile Testing in the XP/Agile Universe, Agile, and XP Conferences, Agile Conference (IEEE): 32-41.

[12] Hu, Y., 2010, The Application and Research of Software Testing on Agile software development, International Conference on E-Business and E-Government (IEEE): 5540-5543.

[13] Menéndez, D. A.; Sousa, T. G. R.; Menendez, E. S. And da Silva, P. C., 2015, Experiment report on the implementation of agile testing, 12th International Conference on Information Technology - New Generations (IEEE): 772-773.

[14] Manzoor, U.; Irfan, J. And Nefti, S., 2011, Autonomous Agents for Testing and Verification of Softwares after Deployment over Network, IEEE: 36-41.

[15] Vatsa, A. And Kumar, S., 2017, Software Production Issues: Analysis and Categorisation, ICACTRP.

[16] Vatsa, A. And Kumar, S., 2016, Software Production Issues and Mitigation Techniques: A Review, IJRRA, Vol. 3, Issue 2, 6-9.

[17] Dearle, A., 2007, Software Deployment, Past, Present and Future, Future of Software Engineering (IEEE).

[18] Leppanen, M.; Makinen, S.; Pagels, M.; Eloranta, V.; Itkonen, J.; Mantyla, M. V. And Mannisto, T., 2015, The Highways and Country Roads to Continuous Deployment, The IEEE Computer Society: 64-72.

[19] Cruzes, D. S.; Moe, N. B. And Dyba, T., 2016, Communication between Developers and Testers in Distributed Continuous Agile Testing, IEEE 11th International Conference on Global Software Engineering: 59-68.