IMPROVING THE REQUIREMENTS ENGINEERING PROCESS THROUGH AUTOMATED SUPPORT: AN INDUSTRIAL CASE STUDY

Fabio Alexandre M.H. Silva, Bruno A. Bonifacio, Fabio Oliveira Ferreira, Fabio Coelho Ramos, Marcos Aurelio Dias and Andre Ferreira Neto

Sidia Institute of Science & Technology, Manaus, Amazonas, Brazil

ABSTRACT

Although Distributed Software Development (DSD) has been a growing trend in the software industry, performing requirements management in such conditions implies overcoming new limitations resulting from geographic separation. SIDIA is a Research and Development (R&D) Institute, located in Brazil, responsible for producing improvements on the Android Platform for Samsung Products in all Latin America. As we work in collaboration stakeholders provided by Mobile Network Operators (MNO) from Latin countries, it is common that software requirements be provided by external stakeholders. As such, it is difficult to manage these requirements due to the coordination of many different stakeholders in a distributed setting. In order to minimize the risks, we developed a tool to assist our requirements management and development process. This experience paper explores the experience in designing and deploying a software approach that facilitates (I) Distributed Software Development, (II) minimizes requirements error rate, (III) teams and task allocations and (IV) requirements managements. We also report three lessons learned from adopting automated support in the DDS environment.

KEYWORDS

Industrial case study, requirement management, DSD, distributed software development, RM, automation, industrial experience.

1. INTRODUCTION

Distributed Software Development (DSD) has been a growing trend as the software industry is experiencing increasing commercial globalization [1]. In this scenario, many companies have been adopting DSD in their software products to accelerate the time to market for new products, better customer satisfaction, and higher product quality [2].

On the other hand, working with distributed teams also face new challenges, particularly in requirements management: communication, software documentation and project coordination [3]. As a means to overcome these challenges, the software industry has sought to automate their process and tasks. In the context of SIDIA, there was the need for tools that are essential for collaboration among team members, enabling the facilitation, automation, and control of the entire requirements management process [4]. However, the existing tools are rarely tailored to the needs of a collaborating group of engineers [5]. Therefore, SIDIA had to develop its own tools that meet the company's needs.

David C. Wyld et al. (Eds): MLDS, NECO, SEMIT, IBCOM, SPPR, SCAI, CSIA, ICCSEA - 2021
pp. 227-236, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.111819

SIDIA is a R&D Institute and a Samsung Company strategic partner, located in Manaus-Brazil, that develops innovative software solutions in various areas, such as machine learning, games, data mining and others related to mobile products. SIDIA is responsible for the development of embedded software and improvements on that Android Platform for Samsung Products in all Latin America. The institute collaborates with Samsung Mobile division, located in Korea, and external stakeholders provided by Mobile Network Operators (MNOs) from other Latin American countries (e.g., Brazil, Mexico, Chile, Peru). For this reason, to meet the demands of MNOs, SIDIA works on a DSD environment. MNOs are the main Samsung clients as relates to the acquisition of Samsung's mobile products. Thus, these stakeholders act as middlemen between MNOs and Samsung, who constantly provide software requirements that need to be implemented into Samsung's mobile products. There are several external stakeholders that present a given MNO in a particular country. For instance, there is a stakeholder in Ecuador who represents all of Claro's requirements in that country. Given that there are many countries in Latin America and each country with several MNOs, the management of all the requirements becomes a difficult process and this could lead to error-prone software products.

In this context, the requirements management process becomes difficult due to the coordination of many different stakeholders in a distributed setting, due to geographic dispersion, language and time zone differences. This has led to a challenge of implementing and validating requirements (e.g., requirement consistency, requirement integration problems and wrongly implemented requirements), which leads to long delays and risks during the software development process.

In order to minimize these challenges, we developed a tool to assist in our requirements management process. In this paper we report the experience in designing and deploying this tool, referred to as Checklist Tool, whose main objective is to facilitate the requirements management process. The Checklist Tool improves requirements testing and validation through integration between systems in the context of DSD. Our results show important improvements in team productivity (e.g., minimizing the time to execute tasks), minimizing error rates (with a reduction in 30% error rates) and task allocation (e.g., one developer can simultaneously do more than one task). We also report the lessons learned from adopting automated support in a DSD environment.

This paper is structured as follows: Section II provides some related works. Section III describes the SIDIA process and the support tool added. In Section IV we present the results achieved by using the proposed tool. In Section V we present the conclusion and propose some future directions.

2. RELATED WORKS

In relation to software engineering, one of the areas mostly affected by a DSD environment is Requirements Engineering (RE). To overcome this difficulty software industry moves to automate the requirements management process [7]. According to [6] DSD requires software tools (management tools, development tools, etc.) to minimize problems such as: geographic dispersion, control and coordination breakdown, communication, team engagement and socio-cultural differences. Moreover, It is important to propose and analyze tools in real scenarios [4][12]. We describe some existing requirements management tools in the following paragraphs.

Sinha et al. [8] proposed a distributed requirement management tool called EGRET (eclipse-based global requirements tool), after interacting for more than one year with approximately 30 IBM employees, involved in distributed development. The EGRET prototype was tested in three

projects at the requirements definition stage. Users reported a good experience: "*found the tool very useful for capturing requirements, having discussions, and tracking requirement changes*". Goda Software presents a solution for requirements management in the form of Analyst Pro [9]. It facilitates requirements specification, tracking and visual traceability analysis. It is a scalable solution, which can provide a collaborative environment that allows sharing of common pool of project information among stakeholders. The requirements can be tracked through design and testing.

Vitech Corporation developed a tool for requirements management, CORE 5.1 [10]. Main features of CORE 5.1 are: reducing schedule risk, improving communication, enabling collaboration, defining and verifying requirements. It also ensures completeness and consistency as well as provides facility to plan tests at an early stage. It also ensures up-to-date documentation and improves planning, visibility and control.

Projectricity developed a requirements management tool called Projectricity [11]. It is a web-based project management platform that enables project team members to efficiently communicate and work collaboratively no matter where they are located. It manages the requirements at all levels of project. It has ability to manage project and task information. Also it is able to manage requirements, test plans, change requests, traceability, and problems in requirements, risks and documentation.

As we can observe, the most common feature in the above-mentioned requirement tools is the issue tracker, some of them include attachment of requirement documents. It is no surprise that issue traceability is very important for requirements management since it is an important component for functionality testing and software validation. In the context of our proposed tool, we combine some features from the related works, including: requirements tracking, requirements specification, risk reduction, communication improvement, collaboration and requirements validation. We describe this in more details in the next section.

3. THE SIDIA MOBILE PRODUCT REQUIREMENT PROCESS

SIDIA develops and updates embedded software for Samsung products commercialized in Latin America. The process of developing and updating this software is divided into three main categories: new models, Operating System (O.S.) upgrade, or maintenance release or MR (for products already in the market). During the software development process, the main objective is the generation of releases. Releases are software versions containing bug fixes, security updates and requirements provided by MNOs. Therefore, once a software version is released, be it a new model, O.S. upgrade or MR, this version goes through a series of tests including validation of MNOs. Once the release is approved, it is then propagated to the respective mobile devices and the end user can download and install. Figure 1 below presents the release process with respect to the requirements management process.

The Android Platform development process starts with requirements definition, by external stakeholders representing MNOs (represented by step (1) in Figure 1 above). The MNOs define, refine and add the requirements on the external system containing information such as O.S. version, device information, mobile applications (apps), wallpapers, and other features (represented as System Requirements in Figure 1).

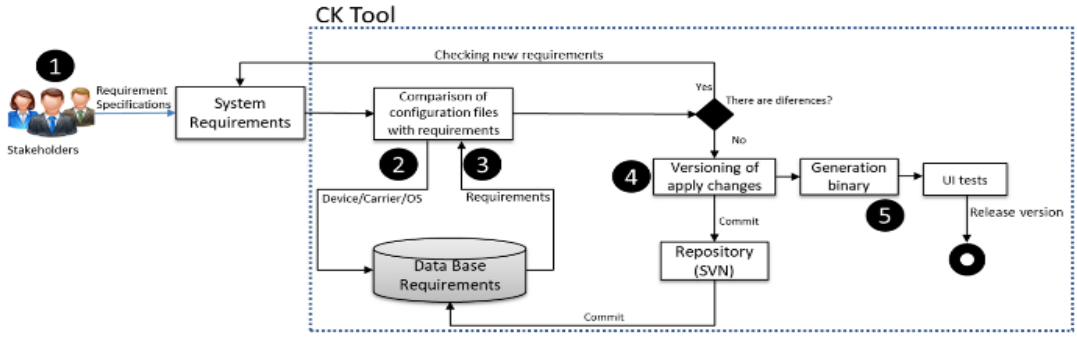

Figure 1. The Release Process from the Requirements Management Perspective.

After that, SIDIA's development team collects requirements and implement according requested by each MNO to each model. This goes through a series of verifications, comparing the requirements from MNOs with that stored in a local data base (steps (2) and (3) of Figure 1). This verification goes on until there are no more differences between the requirements in the database and requirements from MNOs. Upon completion of this phase, the new requirement changes are embedded to the software (step (4) of Figure 1) and stored in a repository. In parallel, the tool is integrated with an SVN (subversion) server and a model compile feature.

system, that uses continuous integration to control file versions and built binaries and tests. Once embedded, a binary is generated (step (5) of Figure 1) and this goes through a series of UI tests. Once the tests are successful, the binary is released.

The main verification step of this process (steps (2) and (3)) has been manually done in the past and this has led to human errors, missing or wrong requirements, applications with wrong versions, which led to long release delays, and rework. Some of these errors have led to serious consequences like delays in market delivery and consequently monetary loss. This led to the need for an automated tool which verifies and applies the requirements with very little human intervention.

For this reason, the Checklist Tool was developed. This solution aimed to automate the requirements validation and testing, minimizing errors occurrence and rework related to missing or wrong requirements. This tool is described in more details in the next section.

4. CHECKLIST TOOL

The tool is divide into three modules: (1) Requirements, (2) Business Intelligence, and (3) Requirement Manager. Initially we developed the Business Intelligence Module to capture activity logs from developers. This feature was important to collect the team's data and create a dataset containing information for each developer. Such information includes: time taken to execute tasks, previously executed or applied MNO requirements, devices to which MNO requirements were applied and average errors committed while executing or applying the requirements. Based on this information, this module is able to recommend tasks to developers.

It is worth noting that even though the tool can recommend tasks to developers, the developers can also manually choose the tasks or tasks can be assigned to them. To use this module, the developers must first authenticate with their ID. Once authenticated, the Checklist Tool can then assign the developer with tasks to apply a software requirement. This is shown in below.

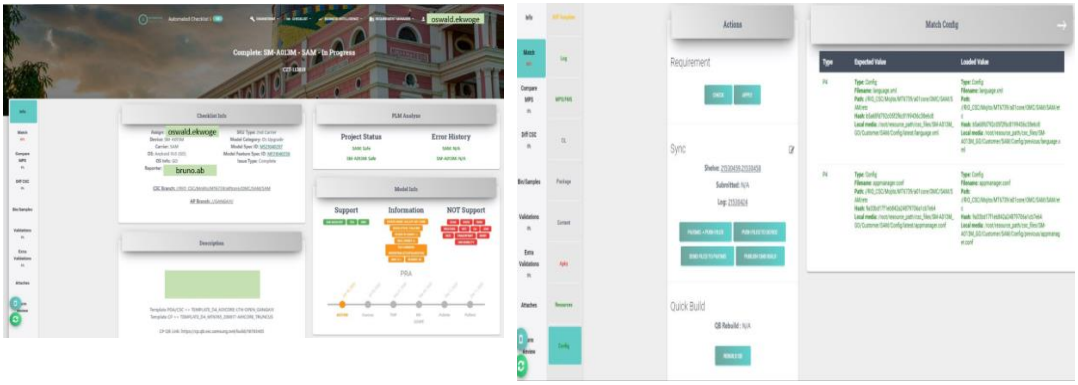

Figure 2. Checklist Tool Dashboard and Requirements Verification and Execution.

As shown in Figure 2.1, the task generated by the Checklist Tool is composed of: developer to whom task is assigned, the device under test (shown as example was the Samsung Galaxy A01 Core (identified by code name SM-A013M)), the MNO, in this case Movistar (we identify the MNO by ID, in this example we used SAM as code for Movistar for Peru), the operating system used (GO - Android GO) and its version (11.0), the model category, in this case, O.S Upgrade. In addition, the Checklist Tool shows the requirements that have to be applied using tags for each requirements comprised of the model specification and feature specification. On the left hand corner of the dashboard, there are certain actions that the developer can choose, such as Match, Compare model specification, Diff on User Interface (UI), Validations, Extra Validations, Attaches and Form Review. When the developer clicks on the Match button, this takes him/her to a new screen where he/she can check if there are new requirements, and in case there are, he/she can apply them (Figure 2.2). On the right hand side of this screen is the Match Config, which compares the expected value with newly loaded value. This is shown when the Check button is clicked. Green text implies expected and loaded values are the same; red means the loaded value is different from the expected value. In this case, the apply button can be clicked to apply the new requirements. After application, the Check button can then be clicked again to do another verification.

In addition, the developer can verify the Device model specifications. This verifies features like power on image, power off image, lock screen image, wallpapers, ring tone, message tone, alarm tone, power on sound and power off sound for both Samsung's and the customer's (MNO's) specifications. This is shown in Figure 3.1 below.

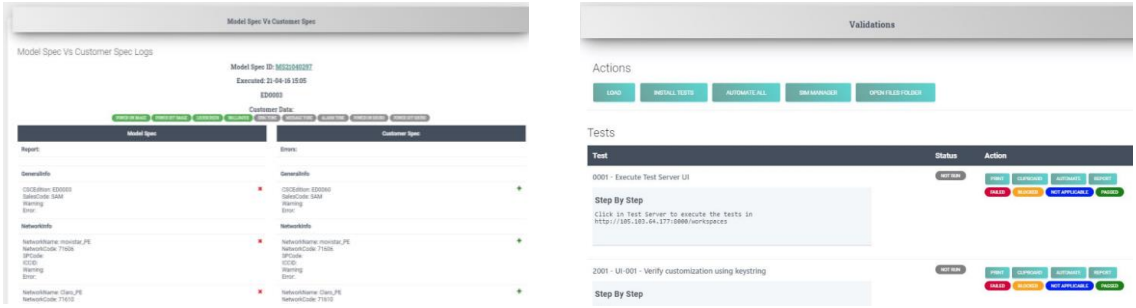

Figure 3. Requirements Validation and Device Model Specification Verification.

As shown in Figure 3.2, the left column shows the Model Spec while the right column shows the customer spec. Once each feature is verified, the color is changed to green; the features still to be

verified are left in grey. In addition to the model spec verification, features are also verified. These features can include application permissions, device permissions, network features and application features. This verification is usually done in an external system and the tool just compares the results with the information in the repository, in this case, Perforce (P4). The tool just compares the differences and this is done using the tool's Diff UI.

Another important feature of the tool is validation. The Validations feature will validate the newly applied requirements. Upon clicking the Load button, all the tests for that particular model and MNO are loaded. The tests are run when the developer clicks the Install Tests button. In addition to displaying the tests, the tool also displays the step by step so that the developer can understand exactly what test is being executed and how it is executed. The tool then displays the test status, which can be: PASSED: all tests were passed; NOT APPLICABLE: when test was not applicable; FAILED – when a requirements was wrongly applied; BLOCKED: when an external situation blocked the test from being executed, for instance, samples or binary available; and ALREADY EXECUTED: when test was performed and approved by previous version. One of the tests involves taking screenshots as evidence. The screenshots are stored under "Attaches" in the tool. After applying all requirements, the tool then stores all modifications and test results in a repository for future use and this cycle continues.

Before this tool was developed, the entire process was done manually. Today, most of the tasks have been automated. There are many other features that have been added to the tool, but have been left out of this work, and many more features are being implemented. We hope to publish this in future works. In the next section, we present some important results obtained by using this automated tool. Furthermore, we also present the results of a survey done with developers about their experience using the tool, as well as some lessons learned.

5. CASE STUDY

After the team started using the tool, two main metrics were evaluated: average time to execute tasks and average errors. The time to execute tasks was calculated based on different task phases which are: time to collect and validate requirement, time to apply requirement, time taken for versioning (embedding software and build generation), time to execute tests and total time taken to execute all these steps. This is summarized in Table 1 below. 3

Table 1. Time taken to Execute Tasks Results.

Type	*Requirement Collection and Validation*	*Requirement application*	*Versioning* *(embedding software + build generation)*	*Test execution*	*Total Time Taken*
Manual	~30 min	~30 min	~2 hours	~5 hours	~8 hours
Automated	~3 min	~1 min	~1 hour	~2:30 hours	~3:30 hours

As can be observed from Table 1, without the tool, developers used approximately 30 minutes to collect and validate requirements; with the tool, it took just approximately 3 minutes. This implies a 90% time gain by using the tool. In terms of application requirement, it took approximately 30 minutes to do this manually, while the tool performed this activity in about 1 minute, implying a 96% time gain. As relates to versioning, it took about 2 hours to perform this activity when manually done as opposed to just about 1 hour when executed using the tool, implying about a 50% time gain. Finally, when the tests were manually executed, it took approximately 5 hours to perform this activity while the tool reduced this time to almost half the time (two and a half hours). It is worth noting that by the time this version of the tool was developed, just over 80% of the tests were automated. The team is currently working to automate

all tests. In total, it took almost a day's work to apply a requirement when done manually, as opposed to just about three and a half hours when performed using the tool. Therefore, by using the tool to execute tasks, we gained about 37.5% of time.

In terms of average errors, we collected data (logs) from 4,500 tasks half of which were manually executed and half were executed using the tool. The task selection and division was random in order to avoid any bias in our results. With respect to manual execution, it was observed that 12% of the tasks presented an issue. On the other hand, with automated tests, about 2% of the tasks presented some issue. This presents about 83% error reduction. These results are very good even though there are several improvements being done on the tool with the aim of achieving near 100% error reduction, especially if all tests can be automated.

In terms of experience of use, we conducted a survey with developers in order to understand their experience using the tool. In total, 36 developers participated (denoted P1 – P36). The survey was a questionnaire composed of just two questions: (i) In relation to the automated tool, how do you classify your experience using the tool, given that 1 is "very bad", 2 is "bad", 3 is "neither good nor bad", 4 is "good" and 5 is "very good"? (ii) Could you describe your experience with the tool in a few words and if possible, suggestions for improvements? Both questions were mandatory even though some participants responded to question (ii) with "I have nothing to say." The results are summarized in Figure 4 below. As can be observed, 15 participants (42%) had a very good experience with the tool, 15 (42%) had a good experience, 4 participants (about 11%) neither had a good nor bad experience, while 2 participants (5%) had a bad experience. Those two participants (P3 and P9) who had a bad experience with the tool respectively explained their reasons and provided suggestions for improvement as follows: "*The tool's buttons and processes are confusing. I will suggest that the UX and usability be improved*.", "*The tool is complete and helps a lot in performing our activities. However, the tool suffers from constant updates which implies the constant execution of a local server by the developer*.".

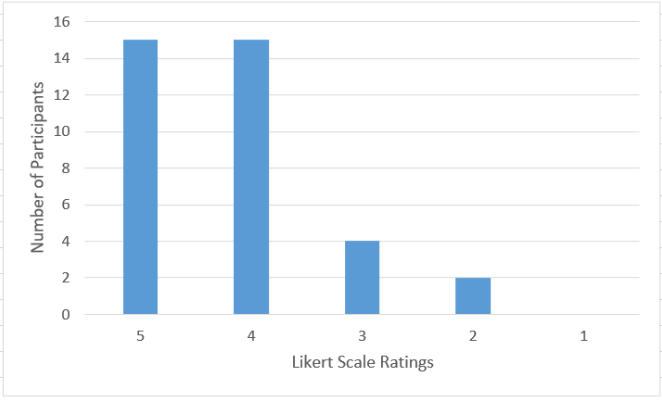

Figure 4. Experience of use results.

Participants (P11, P18, P22 and P34) who neither had a good nor bad experience shared their experiences with focus on UX, transparency and issues faced while working from home during the pandemic: "*The tool has led to a lot of improvements in the process and everything can now be done on a single tool. However, process automation has led to less transparency which is a disadvantage for newly integrated developers*.", "*executing certain tasks when working from home has been an issue, and this has led to some tasks taking longer to execute.",* "*The UX needs to be improved. Certain buttons must have their positions changed as it can be confusing at times.*" "*Improve the execution time for those working at home due to delays caused by the VPN.*" (NB: As relates to VPN, during the pandemic, all teams were forced to work from Home Office

and as such, in order to guarantee protection on Samsung's network, it was necessary for every employee to install Samsung's VPN. As such, employees reported delays in connectivity with several Samsung applications. This also affected the automated tool which had to access several of these third-party applications and hence a delay in executing certain tasks.).

As for participants who had a good or very good experience, some of them reported certain issues with the tool, the main which are: UX, VPN, bug reports. For instance, participant P5 reported that "*I have a very good experience using the tool. However, the tool has presented several bugs. I have the impression that new features are tested in production. If this is the case, I would suggest that a test environment be created in order to avoid bad user experience*.".

In terms of positive aspects about the tool, we classified participants' experiences into categories: execution time, error rate, robustness, standardization and continuous improvement. This is summarized in Table 2 below.

Table 2. Positive aspects of the tool.

Feature	***Description***	***Participants***	***Example***
Execution time	Time taken to execute tasks	P8, P10, P17, P18, P20, P21	P17: "*The tools had greatly reduced the execution time of certain tasks.*"
Error rate	Rate at which errors occur while executing tasks	P24, P27, P29	P24: "*The tools has facilitated the whole process and also greatly reduced the error margin.*"
Robustness	Reliability of results	P2, P10, P27	P2: "*The tool has ensured reliability in the analysis and application of requirements.*"
Standardization	Task execution follows the same standard	P10, P33	P10: "*...I can easily say the process has become more standardized, with quality and rapidity.*"
Continuous improvement	Constant tool updates	P28, P37	P27: "*The tool is always undergoing continuous improvements, this is excellent. Congrats to all involved!*"

Based on these results, there were some lessons learned

2.3. Lessons Learned

Lesson Learned #1: The release process can be considered an "external body of knowledge" built by a set of external stakeholders. In this context, the tool helped to specify everything that is known considering MNO and Samsung requirements.

With the tool, it was possible to provide greater quality in the specification and application of requirements, leading to lower error rates and quicker time-to-market.

Lesson Learned #2: The tool must consider device model characteristics and specific features from MNOs. This information has to be linked aiming to maintain requirement consistency. The historical data can be used to guarantee that requirements do not contain internal contradictions.

The tool has been able to provide standardization between requirements and device model characteristics. This has led to an alignment between the team, MNO and Samsung.

Lesson Learned #3: The tool has become an infrastructure that supports verification, validation, and testing of releases on the device set. It can be supported by using device farms. We started a

simple infrastructure that needs to be refined to consider: more devices connected at the same time and access (with levels of control) by external stakeholders to help validate the application of requirements.

Continuous improvement is important to correct problems related to bugs, time to execute tasks and results reliability. In addition, the tool should be able to simultaneously execute several activities which will lead to even more time gain and even faster time-to-market.

In the next section, we present our conclusion and future works.

6. CONCLUSIONS

As stated by Portillo-Rodriguez [4] and Anwer [12] there is a need for tools that support software engineering processes in the context of DSD and more specially requirements management. Most of the existing tools considers issue tracker as the main feature. Other strategies to manage requirements in DSD scenario have yet to be explored [1]. Our proposed tool, named Checklist Tool, considers three modules: (1) Requirement; (2) Business Intelligence, and; (3) Requirement Management.

Checklist Tool is part of a software tool-based approach that facilitates: (I) Distributed Software Development, (II) minimizing the requirements errors rate, and (III) teams and task allocations. The Checklist Tool integrates external requirement system, and processing to apply requirements and versioning changes, maintain changes history to help developer's team on management applied requirements.

We have important contribution to productivity team. The automated support assists to minimize time to execution tasks, wrong requirements and overload team. However, we realize there is a need to build a supporting infrastructure that allows validation of applied requirements and release testing in a device family. It is important to remember that releases consider features of the Mobile Network Operators (some even cultural), the manufacturer (in this case, Samsung) and the operating system (Android). Another interesting aspect to be investigated is how to minimize the impact of developer misunderstanding of the requirement. In our case we applied the business intelligence module to recommend a set of requirements for certain developer profiles. However, it is an aspect that still needs further investigation.

ACKNOWLEDGEMENTS

The authors would like to thank everyone, just everyone!

REFERENCES

[1] M. El Bajta et al. "Software Project Management Approaches for Global Software Development: A Systematic Mapping Study" Tsinghua Science and Technology, 2018, 3(6):690–714

[2] M. Lormans, H. Van Dik, A. Van Dersen, E. Nocker, A. de Zeeuw, "Managing Evolving Requirements in an Outsourcing Context: An Industrial Experience Report" In: Proceedings. 7th International Workshop on Principles of Software Evolution, 2004, pp. 148 – 158.

[3] G. Kanakis, Fischer, S., Khelladi, D.E. and Egyed, A., 2019, May. Supporting a flexible grouping mechanism for collaborating engineering teams. In Proceedings of the 14th International Conference on Global Software Engineering (pp. 119-128). IEEE Press.

[4] J. Portillo-Rodriguez, A. Vizcaino, M. Piattini, S. Beecham, Tools used in global software engineering: a systematic mapping review, Information and Software Technology (2012).

[5] Akbar, M. A., Sang, J., Khan, A. A., Mahmood, S., Qadri, S. F., Hu, H., & Xiang, H. (2019). Success factors influencing requirements change management process in global software development. Journal of Computer Languages, 51, 112-130.
[6] Akbar, M. A., Shafiq, M., Kamal, T., & Hamza, M. (2019). Towards the Successful Requirements Change Management in the Domain of Offshore Software Development Outsourcing: Preliminary Results. International Journal of Computing and Digital Systems, 8(03), 205-215.
[7] M. Mukhtar, Z. H. Chuhan, Z. Ahmad, Tools for Requirements Management in GSD: A Survey, International Journal of Scientific & Engineering Research, Volume 6, Issue 4, 2010.
[8] V. Sinha, B. Sengupta and S. Chandra "EGRET: A Collaborative Tool for distributed requirements management", TR, report RI06001, IBM Research 2005.
[9] Goda Software - http://www.analysttool.com - Accessed in: 2020.01.06
[10] Vitech Corporation - http://www.vitechcorp.com/solutions - Accessed in: 2020.01.06
[11] Projectricity Project - http://www.projectricity.com/ - Accessed in: 2020.01.06
[12] S. Anwer, L. Wen, Z. Wang, S. Mahmood, Comparative Analysis of Requirement Change Management Challenges Between In-House and Global Software Development: Findings of Literature and Industry Survey. In: IEEE Access (Volume: 7), pp. 116585 – 116611, 2019 Lee, S.hyun. & Kim Mi Na, (2008) "This is my paper", ABC *Transactions on ECE*, Vol. 10, No. 5, pp120-122.