A framework for improving the process of discovering potential errors at the requirements engineering stage

Mohamed A. Hagal
Department of Software Engineering
University of Benghazi
Mohamed.hagal@uob.edu.ly

Fatima Faraj Musbah Saeid
Department of Software Engineering
University of Benghazi
fatma.faraj@uob.edu.ly

2022 International Conference on Engineering & MIS (ICEMIS) | 978-1-6654-5436-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICEMIS56295.2022.9914311

***Abstract* - The process of developing high-quality software depends on the extent to which it meets what is required of it completely and correctly. As a result, the requirements validation process and the testing phase are considered as the most critical stages for ascertaining exactly what the product will offer. Many efforts have been made to prepare methods and techniques to facilitate the testing process and ensure its quality. However, there is a lack of focus on test cases which can lead to potential flaws such as requirements issues.**
As a result, this paper has been working on providing a comprehensive framework that enables software developers to focus on the underlying errors in an organized documentation manner, as well as to be supportive and complementary to the various processes of validation and testing, by focusing on the requirements validation process .some examples are presented in this work to clarify the mechanism of the proposed framework, in which the framework demonstrates a clear mechanism for focusing on potential faults by following requirements in the requirements engineering stage.

Index terms- Software engineering, software requirements, requiermenets validation, software requirements risks.

I. Introduction

Software development has begun to control and organize large areas of our life activities, and this space expands every day. It is necessary to have a way in order to manage software development, from the moment an idea is conceived through the stages of development until it is released (i.e., in order to arrive at a product that meets the requirements of its stakeholders).
Software Development Life Cycle (SDLC) is a sequential process in which to create or maintain software. It includes various stages that start from the stage of requirements elicitation to the stage of software maintenance. There are a wide range of models and methodologies that development teams use to develop software systems, which provides a framework for planning and monitoring the software development from the outset [1][2].
SDLC includes a set of different phases, starting with the functional system requirements, followed by design and implementation phases, then testing comes in. Carefully organization of these phases improves the performance and efficiency of software projects, whereas disregarding inherited errors/bugs between these phases weakens their roles [3][4]. The main steps of the SDLC are depicted in Fig. 1.

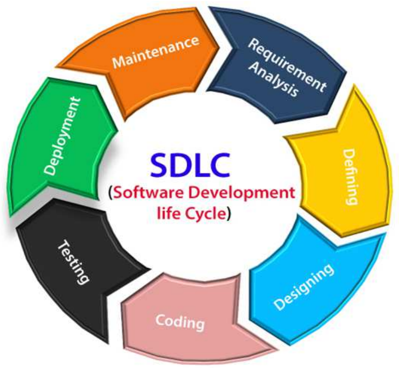

Fig 1 Major phases of SDLC models [2]

It is obvious from Fig. 1, that the requirements engineering stage is the first stage in the SDLC on which the other stages of software development rely. The importance of this stage is to understand the stakeholders' needs, and then document what their software system will do. This stage consists of several activities: requirements elicitation, requirements analysis, requirements documentation, requirements validation, and requirements management. This referred to as the requirements engineering(RE) process [5]. Fig. 2 illustrates the activities involved of this stage.

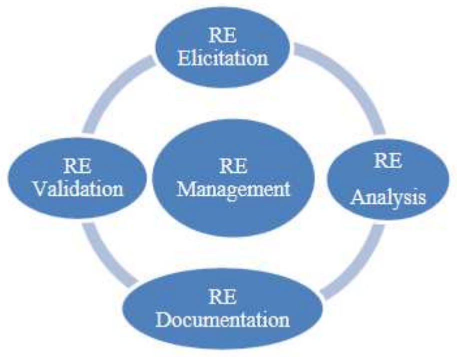

Fig. 2 Requirements Engineering process [5].

Furthermore, the requirements validation activity aims to guarantee that the requirements are correct and accurate, Thus writing the requirements in a clear and unambiguous manner is critical.
On the other hand, as known, the software testing process is a late stage in which the emphasis is on requirements specification and the software operation. This may overlook some inherent errors (potential faults) that testers may not

notice. For example, the inconsistency in requirements can cause such errors. Therefore, by potential faults, we mean errors that the system testers may overlook, with regard to the inconsistency of the main components and subcomponents related or dependent on them, in a manner that ensures their consistency, accuracy and completeness. Therefore, the proposed work aims to improve the software testing process by detecting potential errors at early stages of the SDLC. This can be considered as a complementary process to the software testing stage.

II. Requirements validation activity

The most essential activity in the requirements engineering (RE) stage is requirements validation, consequently, it is necessary to delve a little bit to give an overview of this stage, and thus the role and responsibility of this activity will become clear.

RE stage is the first and most important stage in the SDLC, as it focuses on collecting and understanding the requirements of the stakeholders in an appropriate way for collecting clear requirements about what the software system is expected to perform. As a result, it is thought to have a significant impact on all subsequent stages. Whereas engineering requirements focus on understanding the purpose of the software system to be built and gathering system requirements by collecting and extracting them from the stakeholders. Then thoroughly analyzing and documenting, which leads to building a software system that clearly performs what is required of it. Furthermore, the primary goal of Requirements Engineering is to guide development toward the production of a correct product. Hence, one of the problems with developing clear requirements at the time is the differing views of the stakeholders on them. So, if requirements are not specified properly, the system will cause a lot of problems that will be expensive to repair. Moreover, Effectiveness of requirements validation is the activity responsible for validating the requirements that have been documented, in order to ensure their correctness and accuracy, as well as resolving any ambiguities or inconsistencies therein. After ensuring the accuracy of these requirements, they are documented and forwarded to the next stage, which will focus on designing the system and how it will be built [4]. In addition, The cost of software testing will be reduced if testers are involved from the early stages of the development process [6] [7].

On the other hand, when testing is taken into account at the requirements stage, defects are detected early. As a result, software projects will be more robust in terms of design, implementation, and maintenance. Studies have shown that the goal of adding testing into the SDLC can be summarized as follows: (1) defects that are subsequently discovered have a root cause (i.e., poor requirements), and (2) if the error is detected early, it will not be too expensive to fix (Pandey and Batra, 2013). In conclusion, According to NASA findings, "problems that are not found until testing are at least 14 times more costly to fix than if the problem was found in the requirements phase" [8].

Thus, the requirements stage should include the integration verification process later, in which the components of the software interact correctly with one another, additionally, the design verification process should not neglect checking the consistency of the software architecture and its requirements [9]. Therefore, checking the coupling between components early must be taken in consideration.

Furthermore, detecting requirements conflicts causes problems in the development of software systems, delays their development, and exceeds the proposed cost of producing them. However, the process of conflict detection is critical for verifying requirements, and detecting the conflict is a significant challenge in and of itself [10].

III.REQUIREMENTS VALIDATION TECHNIQUES

The goal of using requirements validation techniques is to ensure the validity of those requirements and document them in a way that ensures the success of the software system later. There are many requirements validation techniques, so a brief look at a few of them will be given [11]:

A. *Inspection*

Inspections are a technique of manually checking requirements in order to ensure that they are correct and meet the needs of stakeholders.

B. *Prototyping*

Prototyping facilitates the process of validating requirements, by attempting to simulate the system to be developed.

C. *Requirements testing*

The objective of this technique is to create test cases for all requirements that are documented in the software requirements specification (SRS).

D. *Viewpoint-oriented requirements validation*

The purpose of this technique is to facilitate the process of extracting requirements, by giving space to the different points of view, and then preparing an approach for negotiating these views in order to resolve the contradictions and ambiguities that surround the generation of valid requirements.

IV. LITERATURE SURVEY

Tsai et al. [12] presented a model of software system development life cycle named as "Test design stages processed model" (TSP), and they stressed that iterative test design stages should be incorporated at each phase of the software development lifecycle. However, a clear mechanism for detecting potential errors has not been explored.

Bose and Srinivasan [13] conducted a study on how to diagnose software errors. Three artificial intelligence techniques were used: "spectrum kernel, SVM, and semantic latent analysis ", and these techniques showed encouraging results. Their focus was on detecting errors during the implementation phase only. This may neglect the potential errors, such as requirements issues and issues of design coupling.

Zheng et al. [14] explained that no single technology can detect all errors. They also pointed out that the techniques for analyzing errors represented in customer reports about the problems that appear in the software. Their study were conducted on three large software systems that were developed in Nortel Networks , and they stated that statistical analysis can be effective for identifying problem modules. Also, it can be considered as a complementary to the other fault-detection techniques. This encourages that stress on potential errors must be included in the testing techniques that undertake to ensure testing of the correctness of the software.

Eichinger et al. [15] conducted a study to discover errors in the software development process. An approach has been presented in order to locate errors that are not predefined or unfamiliar. As for known and familiar errors, the approach increases the accuracy of their identification. They have used graph mining to significantly locate errors. The approach achieved excellent results, but the focus was on the implementation phase. In order to ensure quality of software systems, all stages should be verified.

Tuteja and Dubey [3] presented a study in which they identified a list of testing methods that could be applied at each stage of the SDLC, and recommended testing at each of these stages is necessary. This reinforces the intent of this work, since potential errors are emphasized at an early stage.

Kaur and Singh [16] conducted a study on analyzing and comparing a number of testing techniques, in order to determine which is better at detecting errors. However, their conclusion is that not all errors can be found in software systems. Thus, this leaves open the possibility of further research to improve the testing process.

Dhanalaxmi et al [17] conducted a general study on error detection techniques in an effort to build high quality software, and indicated that there is no general mechanism for testing that is applicable to all software systems. They concluded that, there is a need for testing techniques to help improve commercial software.

Wong et al [18] presented an overview of some software techniques for tracking errors, by conducting a survey of many masters and doctoral studies from 1977 to 2014, and they used the questionnaire to find relevant studies in order to identify possible errors in the techniques used. Most of the studies focused on bugs during software implementation. However, this study did not address the issues of requirements and issues of coupling design.

Yusupbekov et al [19] developed a framework for errors prediction using data mining and intelligent decision support system technique. Associative rules are used in data mining for the traceability of objects hierarchy can be used as a basis of better analysis and gaining additional knowledge to detect and analyze errors. Where they indicated that this work can contribute to the development of requirements for software systems, and to update them in general if the associated rules are well formulated. Nevertheless potential errors still not included.

Creating requirements with excellent characteristics is one of the most important factors that contribute to the success of the program later, and for this reason, efforts have been made, and continue to be made to introduce approaches and methods that contribute to improving the generation of requirements and thus ensuring that they flow well to later stages of development.

Hagal and H.Fazzani [20] introduced an approach aimed to reduce contradictions and ambiguity in software requirements and increase requirements consistency. To capture the degree of requirement inconsistency, use case map (UCMS) is used to visually represent all requirements, and a UML use case diagram is used to represent system functions. The approach did not pay attention to other potential errors such as incompleteness and design coupling issues that may contribute to the software weakness or failure.

Kamalrudin and Sidek [21] presented a review to verify requirements and consistency in order to identify gaps in the requirements specifications. They discussed the different types of techniques used in requirements inconsistency. The models used for semi-formal specifications were discussed. Map representations for searching papers related to consistency determines the technique most commonly used. The presented approach is abstract, and did not address the other issues of requirements such as completeness and necessity.

Gigante et al [22] conducted a study was based on a survey of the basic concepts that must be considered to check the requirements verification, and they proposed an approach to illustrate requirement overlapping. According to the study, there is a great difficulty regarding the completeness of the requirements, and semantic web can be a promising approach for resolving this issue. In order to find methodologies and techniques that solve most of the concerns that may occur in requirements, such as completeness, contradiction, and others, in-depth research is still required.

Stachtiari et al [23] introduced a model-based approach to validate requirements and translate them into system design. In the requirements phase, they used instantiating textual templates, and user defined maps to get unambiguous requirements. In the design phase, the functions of the system were built on the basis of templates of components, and they proposed a phase for checking the design model. They pointed out that the accuracy of the requirements is the foundation for all of the preceding.

Riaz et al. [24] conducted a survey on tools and techniques used to detect ambiguity in natural language requirements from 2003 to 2013. The study revealed the popularity of using these tools and techniques based on citations, and also identified a number of the most important techniques used in this field. This study inspired the work of this paper to present an approach that can help improve or complement the tools that have been proposed.

Yang et al. [25] developed a tool named "Requirements validation through an automatic prototyping" that can automatically detect the inconsistencies in the requirements by generating a number of prototypes that are tested on four case studies where the results are satisfactory, and they concluded that, the tool can be improved later to verify the requirements. This research motivated the work in this paper to improve and develop methods for solving requirements issues, which could help to support any requirement quality improvement process.

Langenfeld et al. [26] introduced a real-time requirement analysis approach aims to transfer the analysis problem to real-time requirements. They translate the formal requirements into an executable program, and then analyze this program as an open source program by using the "ULTIMATE REQANALYZER". They indicated that numerous problems have been identified as serious flows that lead to major problems in subsequent stages of system development. The study did not go into great detail on the testing process, especially with regard to the potential errors that are the focus of this work.

Narizzano et al. [27] conducted a study on an expansion of property specifications patterns (PSOs) that considers the internal consistency of functional requirements, and the results demonstrated that the proposed approach can check specification consistency. They stated that their experiments were carried out on nearly two thousand requirements, and that their future work will concentrate on translating natural requirements into patterns.

Hadar et al. [28] presented an empirical study on requirement inconsistency, taking practitioners' perspectives on it and attempting to identify some dilemmas that contribute to requirement inconsistency. They indicated that the strategies for managing consistency in detecting errors will greatly enhance the consistency process. The research presented in this work aims to improve requirements challenges, rather than just consistency.

Sulaiman et al. [29] investigated the inconsistency between the activity diagram and the class diagram, pointing out that the activity diagram should consistently describe the functions of the class, and emphasizing that the inconsistency occurs when the elements to be described overlap.

Mayr-Dorn et al. [30] presented an approach to assisting and guiding engineers in resolving the inconsistency, and they note that prototypes may contribute to improving the deviation, but they also note that this is rarely addressed in practice. This prompted the emergence of several tools that may contribute to improving this process. As a result, further research into the requirements problems is required.

Guo et al. [10] proposed algorithms for analyzing the characteristics of natural language requirements, and they used heuristic rules to determine a number of conflicts over a number of open requirements datasets. They pointed out that the proposed algorithms gave good results, and that this work requires further investigation.

As previously stated and indicated, most studies dealt with some aspects in a private or abstract manner, in contrast to what was done in this study, which provided an organized framework for tracking the mentioned issues in order to preserve the important characteristics that the requirements must have (Table 1 bellow shows a sample of the available studies and the characterstics that has been addressed, even partially or in an abstract way).

TABLE1
STUDIES, AND CHARACTERISTICS OF THE REQUIREMENTS ADDRESSED IN THEM

Study	**Requirements characterstics**			
	Necessity	**Correctness**	**Completeness**	**Consistency**
(Hagal & H.Fazzani, 2013)				√
(Patel & Gandhi, 2014)				√
(Kamalrudin & Sidek, 2015)				√
(Gigante, Gargiulo, & Ficco, 2015)			√	
(Stachtiari et al., 2018)				√
(Riaz et al., 2019)				√
(Yang et al., 2019)				√
(Narizzano et al., 2019)				√
(Hadar et al., 2019)				√
(Sulaiman et al., 2019)				√
(Mayr-Dom et al., 2021)				√
(Guo et al., 2021)				√
The proposal approach	√	√	√	√

V. THE PROPOSED APPROACH

As mentioned before, potential fault testing is a crucial process in the SDLC that effects the cost and development time of the software. Where the scope of this study will focus on the Requirements stage. At this stage, the work idea concentrates on the validity of the requirements by applying some solutions for testing requirements such as necessity, correctness, completeness and consistency. So the discovery of errors at the early stage reduces the spread of errors during the subsequent stages. The proposed framework is depicted in Fig. 3.

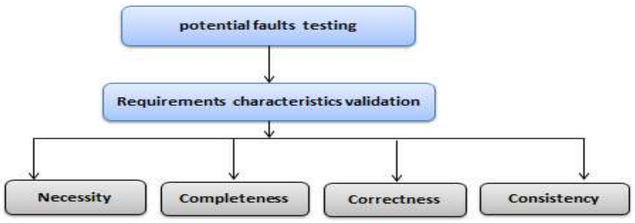

Fig. 3 Overview of the proposed framework

In addition to, requirements' success must take into account that it may pose a significant challenge, especially with regard to validating requirements properties, which if neglected will result in potential errors. Completeness, necessity, correctness, and consistency are examples of these characteristics. Therefore, successful requirements validation is an integrated process for improving potential faults testing. Hence, requirements cannot be considered excellent unless they meet the mentioned characteristics. Therefore, enhancing these characteristics is one of this study's objectives.

A. Necessity

"Necessary" is defined by Merriam-Webster as "so important that you must do it or have it absolutely necessary" (Merriam-Wbster, 2021). Therefore, a requirement to be necessary means that there will be violation in the system's functionality if this characteristic is missed. As previously stated, the requirements stage is an integrated process, so it will be misleading if the requirements engineer puts some requirements which he assume necessary, and therefore he documented them by chance or consider that they are important to the system. While the mistake is to add them without referring to their sources (i.e. stakeholders or documents) [30] .

In addition, determining requirements and knowing their sources is an essential and important process. This process will not be complete without clarity and correctness of these requirements, which is a difficult process that requires more effort to maintain what is mentioned in several researches. The relationship between requirements and their sources is depicted in Table 2.

TABLE 2
REQUIREMENT RESOURCE TABLE

Requirment/Resource	Req1	Req2	Req3	...	Reqn
Source1	X				
Source2			x		X
....					
Source n					

The table, for example, illustrates the sources 1,2, and n, as well as their related requirements, where the character "x" denotes the relationship between the requirement and its resource.

B. *Completeness*

When something is said to be complete, it usually means that no necessary information are missing to be fit for use or serve its intended purpose. The Merriam-Webster dictionary defines complete as “having all necessary parts, elements, or steps” (Merriam-Wbster, 2021). This is consistent with our intuitive understanding. Moreover, for a set of requirements, this means that there are no other requirements necessary for the set of related requirements to fulfill their collective purpose or mission of defining what a given system must be and do, with respect to some specified context[32][33].

Hence, completeness totally depends on the understanding of stakeholders’ requirements and meeting these requirements in the way that they need nothing else.

To confirm the completeness of the requirements, the process of clarifying them is important. So, UML activity diagram with swimlanes is proposed to clarify the requirements, where the interaction between the activities in each requirement, and who are responsible for each zone are illustrated. Fig. 4 depicts an activity diagram with swimlanes.

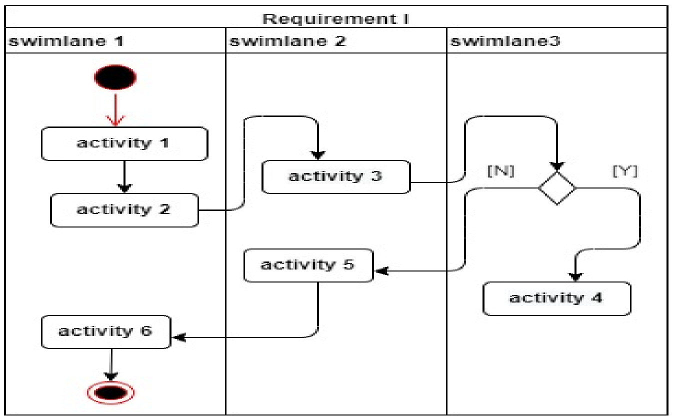

Fig. 4 Activity diagram with swimlanes example.

C. *Correctness*

It means that the correctness of the requirements matches (reflects) the user's real needs, and it emphasizes that they are correct, unambiguous and not repetitive. So, its correctness will affect the part of the program related to it, and then a program is considered correct if it behaves as expected on each element of its input domain [34];[21]. Furthermore, the word “correct” may have many interpretations (i.e. need more clarification) [35]. This gives us the motivation to study the requirements correctness in more depth.

Hence, this characteristic can be considered as a complementary characteristic to the two processes of Necessity and Completeness. So each requirement will be improved if it has been declared as in the Fig. 4 and table 2. Moreover, to complement the necessity table, its validity must be confirmed by the relevant stakeholder.

D. *Consistency*

The process of requirements consistency is regarded as critical in order to guarantee accuracy. This means that no requirements in a specification contradict each other, where all terms have the same interpretation [34]. This requirement uses terms consistently with their specified meanings, so requirement should not contradict with other requirement ,as well as be understood precisely in the same way by every person who reads [36][37].

Requirement descriptions must be complete, with no ambiguity or carelessness for any of the governing conditions. Furthermore, some requirements must be performed after others or have some common parts. Therefore, the dependency between these requirements and the common part in them should be considered and not violated. These parts can be activity(s), conditions, or rules. For example, in order for a student to register his subjects, he must first complete his admission process by considering the required conditions. This means that the process of tracking the dependency of the

Requirement descriptions must be complete, with no ambiguity or carelessness for any of the governing conditions. Furthermore, some requirements must be performed after others or have some common parts. Therefore, the dependency between these requirements and the common part in them should be considered and not violated. These parts can be

activity(s), conditions, or rules. For example, in order for a student to register his subjects, he must first complete his admission process by considering the required conditions. This means that the process of tracking the dependency of the requirements on each other is necessary ,as well as knowing the common parts between them, which should be considered and not violated in any of them. Table 3 illustrates a requirement consistency example.

TABLE 3
AN EXAMPLE OF CONSISTENCY TRACEABILITY TABLE

Requirements				common activity, condition or consistency rule(s)
	Req$_1$	Req$_2$	Req$_3$	
Req$_1$		X	X	
Req$_2$	x			
Req3		x		

The common activity, condition, or consistency rules column in the preceding table illustrates the consistency action(s) that should not be neglected in the related requirements.

In addition, requirements gathering in a complete form will ensure that requirements are excellent and free of violations. Here, are some factors that increases the degree of requirements violations:

- Stakeholders have different views.
- Contradiction between the original (root) and its dependent requirements is not allowed. For example "Req1" must be done before "Req2" and must not violate any of the rules that "Req1" subjected to.

Fig. 5 below clarifies the activities that are common in more than one requirement (i.e., what activities or conditions must be contained in each of the two requirements). The dotted activity shows the common process that must be included in each of the two requirements. For instance, activity "a4" in "requirementi" is the same as the activity "b7" in"requirementJ".

Where the dotted circle in the preceding diagram depicts the consistency activity, which represents the process or rules in both components that should not be ignored(common).

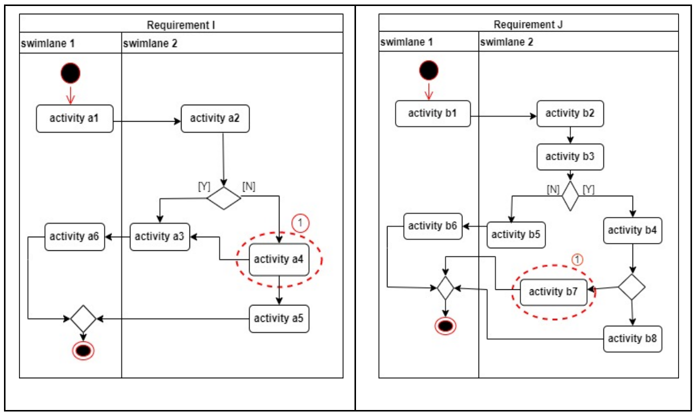

Fig. 5 Activity diagram with swimlane to trace the consistency between requirements

VII. CONCLOUTION

Requirements issues and excessive software coupling are among the most important causes of potential faults that disrupt building successful software. An organized framework has been proposed in this paper that contributes to tracking requirements in terms of complete, correct, consistent, and necessary in order to reduce the resulting errors that may be passed to later stages. Some tracking tables and graphical diagrams have been suggested to make it easier to track these problems. In addition, as high coupling between components is considered undesirable, especially excessive interaction, it is mainly considered as a significant contributor to the propagation of errors between software components, which if not taken into account will lead to week software development. From this point of view, the proposed framework included an organized mechanism for tracking coupling design, where special tables are prepared to track some of the coupling types such as: content coupling, common coupling, control coupling, stamp coupling, and data coupling. From here, it can be concluded, that the proposed framework will support developers in improving their software. Finally, evaluating the results of this framework was carried out based on the preparation of a simplified case study to clarify the mechanism of its work, which we hope would have accomplished the desired result.

As a future work, Applying Natural Language Processing (NLP) concepts can be considered as another trend through which the framework can be improved later by analyzing requirements scenarios and trying to help capture their issues early.

REFERENCES

[1] Y. Leau, W. K. Loo, W. Y. T. Tham, and S. Fun, “Software Development Life Cycle AGILE vs Traditional Approaches,” 2012 Int. Conf. Inf. Netw. Technol. (ICINT 2012), vol. 37, no. Icint, pp. 162–167, 2012.

[2] J. E. T. Akinsola, A. S. Ogunbanwo, O. J. Okesola, I. J. Odun-Ayo, F. D. Ayegbusi, and A. A. Adebiyi, “Comparative Analysis of Software Development Life Cycle Models (SDLC),” Comput. Sci. On-line Conf., pp. 310–322, 2020, doi: 10.1007/978-3-030-51965-0_27.

[3] M. Tuteja and G. Dubey, "A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models," Int. J. Soft Comput. Eng., vol. 2, no. 3, pp. 251–257, 2012.

[4] S. R. Nidamanuri, "Requirements Validation Techniques and Factors Influencing them Santosh Kumar Reddy Peddireddy," 2021.

[5] N. R. Darwish, "Requirements Engineering in Scrum Framework Requirements Engineering in Scrum Framework," no. September, 2016, doi: 10.5120/ijca2016911530.

[6] D. Graham, "Requirements and Testing :," IEEE Softw., vol. 19, no. 5, pp. 15–17, 2002.

[7] B. Lawrence, K. Wiegers, and C. Ebert, "The Top Risks of Requirements Engineering," IEEE Softw., vol. 18, no. 62–63, 2001.

[8] S. K. Pandey and M. Batra, "Security Testing in Requirements Phase of SDLC," Int. J. Comput. Appl., vol. 68, no. 9, pp. 31–35, 2013, doi: 10.5120/11609-6985.

[9] T. Maia and M. Souza, "A Practical Methodology for DO-178C Data and Control Coupling Objective Compliance," pp. 236–240, 2018.

[10] W. Guo, L. Zhang, and X. Lian, "Automatically detecting the conflicts between software requirements based on finer semantic analysis," Inf. Softw. Technol., pp. 1–18, 2021.

[11] H. Anas, M. Ilyas, Q. Tariq, and M. Hummayun, "Requirements Validation Techniques: An Empirical Study," Int. J. Comput. Appl., vol. 148, no. 14, pp. 5–10, 2016, doi: 10.5120/ijca2016910911.

[12] B. Tsai, S. Stobart, N. Parrington, and B. Thompson, "Iterative Design and Testing within the Software Development Life Cycle," vol. 6, no. December, 1997.

[13] R. P. J. C. Bose and S. H. Srinivasan, "Data Mining Approaches to Software Fault Diagnosis," 15th Int. Work. Res. Issues Data Eng. Stream Data Min. Appl., pp. 45–52, 2005, doi: 10.1109/RIDE.2005.9.

[14] J. Zheng, S. Member, L. Williams, N. Nagappan, and W. Snipes, "On the Value of Static Analysis for Fault Detection in Software," IEEE Trans. Softw. Eng., vol. 32, no. 4, pp. 240–253, 2006, doi: 10.1109/TSE.2006.38.

[15] F. Eichinger, B. Klemens, and M. Huber, "Improved Software Fault Detection with Graph Mining," Proc. 6th Int. Work. Min. Learn. with Graphs ICML, no. c, pp. 1–3, 2008, doi: 10.5445/IR/1000008547.

[16] M. Kaur and R. Singh, "A Review of Software Testing Techniques," Int. J. Electron. Electr. Eng., vol. 7, no. 5, pp. 463–474, 2014.

[17] B. Dhanalaxmi, G. A. Naidu, and K. Anuradha, "A Review on Software Fault Detection and Prevention Mechanism in Software Development Activities," vol. 17, no. 6, pp. 25–30, 2015, doi: 10.9790/0661-17652530.

[18] W. E. Wong et al., "Transactions on Software Engineering A Survey on Software Fault Localization Transactions on Software Engineering," vol. 5589, no. November 2014, pp. 1–41, 2016, doi: 10.1109/TSE.2016.2521368.

[19] N. R. Yusupbekov, S. M. Gulyamov, N. B. Usmanova, and D. A. Mirzaev, "Challenging the ways to determine the faults in software: Technique based on associative interconnections," Procedia Comput. Sci., vol. 120, pp. 641–648, 2017, doi: 10.1016/j.procs.2017.11.290.

[20] M. A. Hagal and F. H.Fazzani, "A Use Case Map as a Visual Approach to Reduce the Degree of Inconsistency," Int. Conf. Comput. Syst. Ind., pp. 0–3, 2013, doi: 10.1109/iccsii.2012.6454384.

[21] M. Kamalrudin and S. Sidek, "A review on software requirements validation and consistency management," Int. J. Softw. Eng. its Appl., vol. 9, no. 10, pp. 39–58, 2015, doi: 10.14257/ijseia.2015.9.10.05.

[22] G. Gigante, F. Gargiulo, and M. Ficco, "A semantic driven approach for requirements verification," Stud. Comput. Intell., vol. 570, pp. 427–436, 2015, doi: 10.1007/978-3-319-10422-5_44.

[23] E. Stachtiari, A. Mavridou, P. Katsaros, S. Bliudze, and J. Sifakis, "Early validation of system requirements and design through correctness-by-construction," J. Syst. Softw., vol. 145, no. September 2017, pp. 52–78, 2018, doi: 10.1016/j.jss.2018.07.053.

[24]Riaz, M. Q., Butt, W. H., & Rehman, S. (2019). Automatic Detection of Ambiguous Software Requirements: An Insight. 5th International Conference on Information Management, ICIM 2019, March, 1–6. https://doi.org/10.1109/INFOMAN.2019.8714682

[25] Y. Yang, W. Ke, and X. Li, "RM2PT: Requirements validation through automatic prototyping," Proc. IEEE Int. Conf. Requir. Eng., vol. 2019-Septe, pp. 484–485, 2019, doi: 10.1109/RE.2019.00067.

[26] Langenfeld, V., Dietsch, D., Westphal, B., Hoenicke, J., & Post, A. (2019). Scalable analysis of real-time requirements. Proceedings of the IEEE International Conference on Requirements Engineering, 2019-Septe, 234–244. https://doi.org/10.1109/RE.2019.00033

[27] Narizzano, M., Pulina, L., Tacchella, A., & Vuotto, S. (2019). Property specification patterns at work: verification and inconsistency explanation. Innovations in Systems and Software Engineering, 15(3), 307-323.

[28] Hadar, I., Zamansky, A., & Berry, D. M. (2019). The inconsistency between theory and practice in managing inconsistency in requirements engineering. Empirical Software Engineering, 24(6), 3972–4005. https://doi.org/10.1007/s10664-019-09718-5

[29] Sulaiman, N., Ahmad, S. S. S., & Ahmad, S. (2019). Logical approach: Consistency rules between activity diagram and class diagram. International Journal on Advanced Science, Engineering and Information Technology, 9(2), 552-559.

[30] R. Saavedra, L. Ballejos, and M. Ale, "Software Requirements Quality Evaluation: State of the art and research challenges," pp. 1850–2792, 2013.

[31] Mayr-Dorn, C., Kretschmer, R., Egyed, A., Heradio, R., & Fernandez-Amoros, D. (2021, May). Inconsistency-tolerating guidance for software engineering processes. In 2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER) (pp. 6-10). IEEE.

[32] R. S. Carson et al., "5.1.3 Requirements Completeness," INCOSE Int. Symp., vol. 14, no. 1, pp. 930–944, 2004, doi: 10.1002/j.2334-5837.2004.tb00546.x.

[33] J. Marques and S. Yelisetty, "An Analysis of Software Requirements Specification Characteristics In Regulated Environments," Int. J. Softw. Eng. Appl., vol. 10, no. 6, pp. 1–15, 2019, doi: 10.5121/ijsea.2019.10601.

[34] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," 2003.

[35] G. Gigante, F. Gargiulo, M. Ficco, and D. Pascarella, "A semantic driven approach for consistency verification between requirements and FMEA," Stud. Comput. Intell., vol. 616, pp. 403–413, 2015, doi: 10.1007/978-3-319-25017-5_38.

[36] I. Sommerville, Software Engineering, 9th ed. 2011.

[37] S. Acharya, H. Mohanty, and C. George, "Domain consistency in requirements specification," Proc. - Int. Conf. Qual. Softw., vol. 2005, no. 9, pp. 231–238, 2005, doi: 10.1109/QSIC.2005.24.