NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Wee Kim Wee School of Communication and Information

Manickam Ramalingam Priyanka (G1902176C)
Ramakrishnan Sai Supraja (G1901725C)
Vedapudi Kumar Karthick (G1901762D)

Quality Assurance of Requirement Analysis Phase in the Ariane 5

CI 6204 Software Project Management

AY2019-2020

Abstract:

Quality Management is one of the major factors which can influence the success of any project. Of all activities related to quality management, we focused on quality assurance at the early stage of the software development life cycle which helps in reducing the cost and effort at the latter stages (Noorin & Sirshar, 2017). There can be many reasons for failures in software systems such as lack of adequate testing, poor software design etc (Martin & Niger, 2002). But above all these, requirement document with defects would impact all the stages of the software development cycle and cost for this would be very high. So, it is very important to assure the quality of the requirements (Ejaz et al., 2010). For the high-quality requirement document, quality must be ensured while analyzing requirement document. From the literature study, it is observed that there are multiple factors which determine the quality of the requirement document analysis. We have proposed a model to assure the quality of the requirement analysis phase. Using the proposed model, we analyzed the explosion of Ariane-5 rocket. Ariane-5 is part of the Ariane rocket family, a series of high lifting ELVs (Expendable Launch Vehicle) able to carry heavy payloads to low-Earth orbit and geostationary transfer orbit. After successful launch of Ariane-4, European Space Agency (ESA) designed a heavy lift rocket which is completely new from its earlier models and was called as Ariane 5. It was focused to carry a heavier payload than Ariane 4, predecessor of Ariane 5 (The Economist, 1996). Through this analysis, a conclusion was drawn that failure to capture requirement changes due to inefficient verification of requirement document, lack of collaboration between various stakeholders while preparing and analyzing requirement document had contributed to the failure of Ariane-5.

1. Introduction:

Quality requirements most strongly drive the architecture of software-intensive systems. They play a vital role in determining the success or failure of a mission-critical system (Firesmith, 2005). Quality assurance helps in checking if the requirements meet the desired quality attributes like adequacy, completeness, consistency, etc. A better-quality assurance check in the early stage of software development life cycle (SDLC) helps in reducing the cost effort in the latter stages (Noorin & Sirshar, 2017). The poor quality of a product is due to the result of an ambiguous, unverifiable, incomplete and inconsistent set of requirements (Firesmith, 2003a).

Ariane is a series of European launch vehicles used for space launch. Improved versions of Ariane were launched with modifications in design and with higher commercial payload and improved quality. In the year 1996, June 4, Ariane 5 was designed as the standard launch vehicle for carrying a commercial payload with higher degree of accuracy mission. This project encountered a failure and the rocket exploded 37 seconds after the take-off (Lions, 1996). In spite of the fact that Ariane 5 explosion took place due to multiple factors, incorrect requirement analysis of the project is found as the root cause of the system and software validation failure.

Requirement analysis is the phase of gathering the requirements and analyzing them to produce a better product. This is the first stage of SDLC (Ejaz et al., 2010). Understanding the requirements is an integral part of information systems and it is very important to identify and integrate the needs of various stakeholders (Maguire & Bevan, 2002). Preparation of requirements document would thus impact subsequent phases if not executed properly (Ejaz et al., 2010). Ariane 5 is a suitable case study as it is clearly the result of incorrect analysis of changing requirements and illustrates the quality assurance of requirement analysis.

In line with this, the objective of this paper is two-fold. Our first objective is to propose a quality assurance model for the requirement analysis phase. In spite of the numerous numbers of quality assurance models already available, a general framework is needed to best analyze and amalgamate various factors discussed in other frameworks. This is done to highlight the importance of requirement analysis as the crucial stage. Our second objective is to then use the proposed model to analyze the Ariane 5 explosion.

The structure of the rest of the paper is that, firstly, we will look into the existing literature on the quality models in the requirement phase and summarize the points with a proposed model. Next, the case study of Ariane 5 will be illustrated in detail. Finally, this case study will be analyzed using the proposed model and will be summarized with a conclusion.

2. Literature Review:

Existing literature has shown that the quality of requirement specification can be affected by multiple factors. Here, we will discuss the three main factors which can significantly impact quality. Finally, we propose a conceptual model based on these factors.

2.1 Checklist of Quality Metrics:

2.1.1 Quality Requirements Checklist (Donald Firesmith, 2005)

This paper provides a brief checklist of questions considered during requirements identification and analysis by quality team based on which they can evaluate the requirements. This paper emphasizes that while analyzing, the same technique should not be used to analyze all the requirements. For example, use-case modeling can be used to analyze functional requirements, but cannot be used to analyze safety or performance requirements.

This paper suggests evaluating the checklist to ensure whether the checklist covers the criteria to assess the quality of the requirement. It evaluates the checklist based on below aspects:

- Is each quality requirement describing system-specific quality factor?
- Is each quality requirement verifiable?
- Is each quality requirement measurable to a threshold?

2.1.2. Checklist based on the Goal Question Metric method (Knauss & El Boustani, 2008)

This model emphasizes the creation of a checklist based on certain goals that can be used to analyze the requirement quality. It lists the set of metrics to be assessed to assure the quality of the requirements under two terms. First is formal requirement quality which refers to verbalization rules. For example, a critical typo is a goal under which metrics such as grammar, rules of expression are assessed. However, they reflect the quality of the SRS's content indirectly. The second is content-related requirements quality which refers to goals that need interpretation to some extent. For example, classifying the requirements as functional, non-functional categories. The drawback of this model is that it doesn't have any process to compare functional and non-functional requirements which is important to improve the quality of the requirement document.

2.2 Integrating Requirements based on viewpoints of the stakeholders:

2.2.1 An Experience-Based Approach for Integrating Architecture and Requirements Engineering (Paech et al., 2003)

The purpose of this paper is to provide an approach for integrating both the architectural options design (AOs) and acquisition of the requirements (Functional(FRs) and Non-functional(NFRs)) mutually. The requirement specification is prepared by elicitation and documentation of the problems. The architectural design should give a high level of abstraction for the solution to the requirements.

This paper helps in explaining elicitation, requirement specification and designing of architecture by providing a rationale, predefined architectural patterns, checklists and questionnaires that have been followed in the preceding successful projects. It also discusses the fundamental issues faced during the integration of the architecture, requirements and how this approach overcomes the issues.

The issues are as follows:

1. Different views of different stakeholders during the elicitation of requirements and architectural design options.
2. Identification of the interdependencies between different requirements.
3. Assessment of how different architectural options are specific to a set of requirements.
4. Identification of how past experience supports the integration of architecture and requirements.

This approach discusses the above-mentioned issues in four different modules.

Elicitation: During the elicitation, the stakeholder has to prioritize the quality attributes at a higher level of abstraction for software development. The high-level quality attributes are filtered via the checklist and the questionnaire. The checklist is distinguished to different types based on the

refinement aspect which are the problem, solution refinement and dependencies between quality attributes.

Specification: During the specification, the NFRs will be specified in the requirements document. The checklist helps in completing this activity. A predefined template is used to describe the different NFRs at different places in the requirements document. The NFRs are preferably explained in the use case description. The dependencies are captured by using the refinement graphs.

Design: During the design, the requirements with impact in the architectural design is chosen. In iteration, the architecture is refined based on the refined requirements. After this, the architecture is imposed with the non-functional features.

Experience capture: In this step, the experiences that are collected and consolidated during the predecessor projects used to improve the refinement graphs, checklists, and questionnaires. For this purpose, a set of statements is prepared which involved the processes in software development. After the statements are formed, the software quality experts would select more reliable statements by considering the high-level and low-level quality attributes.

Based on this approach, the FRs, NFRs, and AOs are integrated by means of questionnaires, checklists, refinement graphs, patterns, and rationale. The quality attributes are used to keep the level of abstraction. But the drawback of this approach is, it does not perform any verification and validation after the integration is achieved which is necessary to improve the quality.

2.3 Requirement verification and validation:

2.3.1 Role of Requirement validation in requirement development (Pandey et al., 2012)

Validating and verifying user requirements involves the end-users accepting the specified system with respect to tasks, goals and functionalities. This is usually done by performing requirement reviews with stakeholders. Usually, verification of requirements needs to be done against the system level requirements and Software Requirement Specification (SRS) document. The requirements need to be logged to allow stakeholders communication and succeeding maintenance of requirements.

The requirement specification document acts as the cornerstone for the entire system functionality. The implications of incorrect and inadequate specifications can have severe impact on the entire project. It is very important to check the correctness, consistency, understandability, and unambiguity in the requirement specification and this step is often referred to as validation of requirements. The rationale behind the decisions should be recorded properly to ensure the verification and validation for the subsequent phases of the system development.

Also, it is equally important to analyze that the requirements are stated correctly. This is referred to as verification. Raw requirements are the ones that have not been analyzed and a well-documented notation is not specified. When the entire requirements are stated and described, different stakeholders have to agree upon to these. Hence, it is important to validate these system requirements against the raw requirements and ensure that they are consistent and correct.

Further to this, requirement traceability is an important factor to be maintained throughout the entire process. Traceability is mainly used to generate an audit trail between the requirements and the finally tested code.

2.3.2 An integrated system for end-to-end traceability and requirements test coverage (Ooi, 2014)

Traceability among requirements helps in managing dependencies and in ensuring that all the requirements for the system are completely tested in the test protocols. It is used to identify conflicting requirements and ensure that they are properly verified before moving to further processes.

Traceability is mainly used for its ability to reuse. This enables to share the requirements without any duplication. Bidirectional traceability is one of the good practices, which means that it can be traced in both forward and backward directions.

- Forward traceability – trace the requirements from forward to test. It indicates the completeness of subsequent implementations.
- Backward traceability – trace each test back to its associated requirements. This ascertains that the requirement is kept correct and helps to avoid unneeded tests.

3. Proposed Model:

The following model is proposed by eliminating the drawbacks of the models in the literature review.

3.1 Generate checklist of metrics and goals:

This is the primary step of the requirement analysis phase of our proposed model. In this step, the checklist of metrics and goals are properly generated so as to assess the quality of functional and non-functional requirements.

- **Functional Requirements:** These usually specify the behavior of the product under specific conditions. Description of user tasks and system dependencies may also be mentioned in these requirements. It clearly defines what the system is supposed to accomplish. (Wiegers & Beatty, 2013).
- **Non-functional Requirements:** Certain requirements like operational, security requirements fall under non-functional requirements. They are often referred to as 'quality attributes' of the system, 'quality goals' or 'technical requirements'. Quality attributes like reliability, traceability, security and usability of the product are included in these types of requirements.

Here, the checklist is classified broadly based on two terms.

1. **Formal requirement quality:** This refers to the verbalization rules that are followed when assessing SRS. It includes completely specified process words, neglecting the incomplete comparisons, etc. They can be easily measured and is an objective metric. (Knauss & El Boustani. 2008)
2. **Content related requirements quality:** This refers to the goals that require interpretation. For instance, quality aspects is needed to check if SRS contains a quality model or not and to check if it is sufficiently detailed. (Knauss & El Boustani. 2008)

3.2 Integration of requirements based on viewpoints of stakeholders:

A group of experts should be set up by the Quality Assurance personnel to create a requirement document. The personnel should make sure that certain factors like clarity, redundancy, completeness, and testability are maintained. The implementation of each requirement has to be thoroughly checked. The group of expertise should include analysts eliciting requirements and the design architects of the system. (Ejaz et al., 2010).

The collaboration of the different types of specialists would share knowledge combining different viewpoints like functional and non-functional requirements and architecture of the system. During this stage, the conflicts between different stakeholders are addressed and the architectural decisions are captured. For example, reused requirements are rejected from the final architecture of the component if it is no longer needed. (Paech et al, 2003)

3.3 Verification of the requirements document:

The requirement analysis phase consists of two different kinds of outputs, SRS and Interface Requirements Specification. These documents constitute the complete software development discipline. The Quality Assurance team should properly verify and validate the requirements document. The verification process includes all the requirements agreed by all the stakeholders. If any of the requirement has been proved to be invalid then it has to be marked as an invalid requirement (Ejaz et al., 2010).

The quality assurance team should also ensure that the requirements documents are verified and validated based on the quality factors. It is highly important to check and ensure quality assurance in this phase as it reduces the errors and saves a lot of time, effort and cost.

Finally, once these three steps are combined, the document has to be verified and scanned. If the requirements are satisfied, the development phase is carried out. If not, the whole process has to be repeated once more.

4. Case Description:

4.1. Organization Background:

Ariane-5 is part of the Ariane rocket family, a series of high lifting ELVs (Expendable Launch Vehicle) able to carry heavy payloads to low-Earth orbit and geostationary transfer orbit. Ariane was established and operated by European Space agency, considered as Europe's gateway for space exploration and later transferred control to Arianespace, a satellite launch company who currently manages the marketing and launching of the vehicle (Russo, A. 2011). European Space Agency launched the Ariane 1 at the end of 1979. After successful launches of Ariane 1,2,3 and 4 models, ESA designed a heavy lift rocket which is completely new from its earlier models and is called as Ariane 5. It was focussed to carry a heavier payload than Ariane 4, predecessor of Ariane 5 (The Economist, 1996).

4.2. Project Overview and the intended outcome:

European Space Agency (ESA) developed Ariane 5 to carry a heavier payload than its predecessors. Though Ariane 5 was developed to replace its predecessors, decision had been taken by the Ariane team to re-use the software used in Ariane-4, which was one of the most successful launches of ESA (Lions, 1996). Ariane 5 has been designed to launch on June 4, 1996 for carrying the payload called Cluster Spacecraft, a constellation of four spacecraft. This mission is intended to place the spacecraft in high earth orbit and carry out research into the magnetosphere of Earth and its interaction with solar wind. (Krebs, 2019).

4.3. Timeline of events and outcome of the project:

After successful launches of Ariane-4, further technological development is needed to the Ariane's continuing success. The ESA proceeded with the Ariane 5's development in 1987 and the launcher had been designed by early 1994. Ariane 5 is a two-stage heavy class booster rocket and it consists of three basic parts, the Vulcain propulsion system, the main cryogenic stage and the solid rocket boosters.

On 4th June 1996, at the Guiana Space Centre, Europe's spaceport, the countdown for the first flight of Ariane-5 had started (European Space Agency, 1996). The launch was put on hold at this point because at the time the strict weather criteria for a launch were not met. After a while, favourable weather conditions were observed on the day of launch and there was no chance for the lightning, so the electric field was also negligible. The tracking system that predicts the range and safety requirements were checked thoroughly. The solid boosters and the Vulcain engine were normal, as was the subsequent lift-off. Until 36 seconds after lift off, the flight had a nominal trajectory as listed in the event timeline Table 1. But shortly thereafter both active as well as backup inertial reference system got shut down. This caused unsustainable stresses on the rocket causing the launcher to deviate from its flight path. This triggered the self-destruction of the launcher which is in accordance with specification (Lions, 1996).

After the failure, the Director General of European Space Agency organized an independent Inquiry Board to determine the detailed analysis and root cause of the failure. The board concluded that the Ariane 5 explosion was caused by the loss of signals such as attitude and guidance information after 37 seconds of the lift off. The reason for the information loss was the error in requirement specification and design of the inertial reference system (SRI) (Lions, 1996).

The attitude and trajectory of the Ariane 5 are measured by a software controlled Inertial Reference System (SRI). Then, SRI uses the measured information to guide the engines to maintain the attitude and the navigation. In the failure scenario, the significant technical issues are the software error happened in the Inertial Reference System (SRI) while converting the variable, Horizontal Bias (BH), one of the variables related to the Horizontal Velocity, and the lack of exception handling during occurrence of such kind of software error. The horizontal bias variable is used to change the velocity and the alignment of the rocket (Lions, 1996).

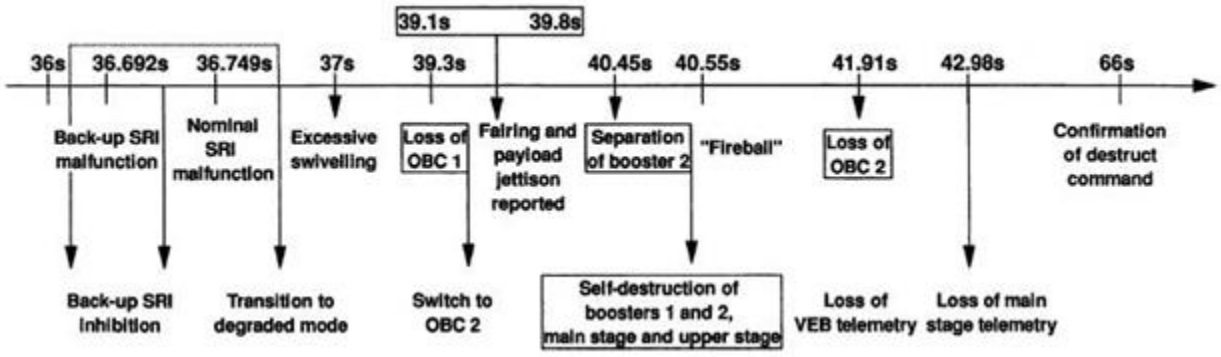
In the design, the variable BH is represented as a 16-bit integer. This is based on the non-functional requirement which was directly taken from design of Ariane-4. Due to the changing requirement of Ariane-5, after lift-off, the value of the variable reached a larger value which can only be represented by 64-bit floating point. Error occurred when the software code tried to fit a 64-bit floating point number (big value) in a 16-bit integer (smaller value) space (James Gleick, 1996). Because of the software error, both active and backup inertial reference system had shut down which is in accordance with the design. As SRI was inoperative, engines lost its attitude and navigation information and rocket changed its path abruptly which triggered the self-destruction

mode as shown in Figure 1 (Lions, 1996).The failure harmed the continuous success record of the ESA rocket family and the total cost lost for ESA is estimated to be $370 million (James Gleick, 1996).

Table 1: Event timeline - Ariane -5 launch (Lions, 1996)

TIME	EVENT	DESCRIPTION
L-11:43:00	Countdown Initiation	Favorable weather conditions were observed on the day of launch. And there was no chance for the lightning, so the electric field was also negligible
L-10:23:00	Filling the stage	The event of power up of the engine was held. During this time, the electrical checks, fuelling configurations were checked
L-06:00:00	Software Load	The program simulation was implemented to increase the capacity of the application.
L-03:43:00	First and second stage propelling loading	In order to generate thrust, the loading was implemented
T-00:00:00.2	Visibility and tracking conditions	The tracking system that predicts the range and safety requirements were checked thoroughly.
T+00:00:07.30	Lift off	Lift off is a normal take-off without a risk. The take-off was successful

Figure 1: Event timeline – Accident (Lions, 1996)



5.Analysis:

In this section, the Ariane-5 explosion case study will be analyzed based on the model proposed in the literature review.

5.1. Insufficient Checklist to assess quality:

A system test should test the interfaces of the system and the whole system (Lions, 1996). In the case of Ariane-5, SRI computes the alignment of the rocket after launch based on the value of horizontal velocity and sends the information to the main computer. After launch, Horizontal velocity exceeded high above its intended value which caused SRI to shut down. This was the main cause of this explosion.

We see that there are two quality metrics which was missed in the quality checklist of the requirement document.

- Coverage
- Verifiability

Coverage:

It was decided by the Ariane team that it was not necessary to check whether the large value of the horizontal bias, one of the variables related to horizontal velocity affects the SRI or not. This check was performed for other variables of the software but omitted for variables related to horizontal velocity (Lions, 1996).

This shows that, in requirement specification document, operational and safety requirements are not properly covered for all variables of the alignment software as well as its impact on SRI operation.

Verifiability:

It was not feasible to do black-box testing for SRI in real-time due to test environment reasons. However, some ground testing by injecting simulated signals of the flight was feasible but it was not performed. After the explosion, the inquiry board which analyzed the causes of the failure had performed such tests with the software of the SRI. With the simulated environment, which includes trajectory data of flight, the inquiry board reproduced the failure scenario of the flight which caused the explosion (Lions, 1996).

This shows that ground tests with trajectory data of the flight would have exposed the failure of the SRI system beforehand. Testers failed to do these tests as the specification document didn’t say so. The SRI requirement specification document did not have flight trajectory data as part of

functional requirements (Lions, 1996). In this case, we see that there are dependencies to achieve the goal of doing system testing of SRI.

For example, “System testing should be performed to ensure system safety.” is just a statement. But it is important to analyze the dependencies needed to perform the testing. In other words, it is important to define in the requirement document whether a certain goal is achievable or not and what is needed to achieve the goal.

5.2 Inefficient integration of requirements based on viewpoints of stakeholders:

The functional and non-functional requirements and architectural design should have been developed in a properly integrated manner. All of the entities are highly interconnected and depend on the stakeholder’s viewpoints. Concentrating on the first and second elements and doing the third element separately is not sufficient enough to accomplish the complete requirements. In order to put a satellite on the specific orbit, the corresponding rocket should follow some projectile path. This requirement shall calculate the corrections based on the actual path and the planned path of the rocket. Before the rocket is launched, the navigation system should also calculate the originating position of the rocket. To compute this, a variable horizontal velocity must be considered. This is very small before the launch and extremely large after the launch. So it has to be computed in both cases. It has become a non-functional requirement. After the requirements are taken into account, the architectural design was done by the designer and the developer for these requirements, which then lead to a change in the requirements that the inertial reference system will compute the corrections of the path of the rocket. Before the launch, the alignment subsystem will provide the initial offset position of the rocket to the inertial reference system (Paech et al., 2002).

However, the computation time taken by the alignment subsystem is high and initialization takes about 45 minutes. To overcome this 45 minute penalty, a new functional requirement is proposed to resume the count down after a few minutes of stoppage. This resulted in another functional requirement and new architectural design which is the system should proceed with the computations for a restricted time of 50 seconds after committing the initial position value. So the alignment subsystem takes an additional functionality. Here the functional, non-functional requirements and the architectural design are carried out in a non-linear and iterative manner without a common viewpoints of stakeholders (Gotel & Finkelstein, 1994).

5.3 Inefficient verification of requirement document:

Matching the system requirements against raw requirements was not done properly. It was evident from reports that Ariane 5 requirements were different from the previous models of Ariane. However, the piece of alignment code that lead to the failure of Ariane 5 was not required in this

project. This piece of code remained operational without fulfilling any traceable requirements. Proper requirement verification would have prevented the occurrence of failure (Nuseibeh, 1997).

The ambiguity in the requirements would have been exposed if the history of requirements evolution, explanations and justifications of the requirement document had been verified properly (Gotel & Finkelstein, 1994). This means, the requirement verification and validation were inadequate. SRI failure would have been easily exposed if the acceptance testing or review had been performed properly along with ground tests. Organized and upgraded project management would have provided an efficient organizational process to analyze the changing requirements on system functionality (Nuseibeh, 1997).

Furthermore, the inconsistencies in the verification of changing requirements of Ariane 5 shows that the requirement traceability mechanism was not effectively implemented during verification. The bidirectional traceability was not traced appropriately. Forward tracing would have been very useful in this case because it is used to trace the requirements from the beginning (collection of raw requirements) to the testing phase. The inconsistencies would have been identified easily if this traceability mechanism was implemented (Ooi et.al, 2014). The disaster of Ariane 5 is clearly due to the inefficient verification of the requirement documents.

6. Conclusion:

The proposed model presents a quality assurance processes which involves preparing a list of quality metrics, analyzing the quality metrics based on the view of requirement analysts and architects to produce the clear requirement document and verifies the requirement document based on requirement traceability. Based on the proposed model, Ariane 5 explosion is analyzed. We conclude that though software error led to the explosion of the Ariane 5, failure to capture the changing requirements (i.e., not taking viewpoints of the stakeholders while preparing requirement document and inefficient requirement traceability mechanism) is the major reason which caused the software error in the first place thereby leading to a poor quality product which ultimately resulted in the failure.

References:

1. Knauss, E. & Boustani, C.E. (2008). Assessing the Quality of Software Requirements Specifications. *IEEE International Requirements Engineering Conference*, 341-342.
2. Donald G. Firesmith (2005). “Quality Requirements Checklist”.*Journal of Object Technology*, 4(9), pp. 31 – 38.
3. Russo, A. (2011). Europe's Path To Mars: The European Space Agency's Mars Express Mission. Historical Studies in the Natural Sciences, 41(2), 123-130.
4. Gunter Krebs (2019). "Cluster 1, 2, 3, 4, 5, 6, 7, 8". Gunter's Space Page. Retrieved 14, October 2019 from URL https://space.skyrocket.de/doc_sdat/cluster.html
5. Alsultanny, Y., & Wohaishi, A. (2009). Requirements of Software Quality Assurance Model. Second International Conference on Environmental and Computer Science, 19–23
6. Noorin, U., & Sirshar, M. (2017). Quality Assurance in Requirement Engineering. Global Journal of Computer Science and Technology – C: Software & Data Engineering, 17(1), 2-5.
7. Javed, A., Maqsood, M., KhurramAshfaqQazi., & Ali Shah, K. (2012, March 31). How to Improve Software Quality Assurance in developing countries. Advanced Computing: An International Journal, 3(2), 17-28.
8. Sommerville, I. (2011). Software engineering (9th ed.). Boston: Pearson. 344-468.
9. Lions, J. L. (1996). ARIANE 5 Flight 501 Failure Report by the Inquiry Board: Failure Report. Retrieved 04 September, 2019, from https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf
10. Nuseibeh, B. (1997). Ariane 5: Who Dunnit?. Institute of Electrical and Electronics Engineers Software, 14(3), 15–16.
11. Launch of Ariane 5 ... an elephant explodes (1996). The Economist, 339(7969), 87.
12. Ariane 5 Explosion Caused by Faulty Software. (1996). Phillips Business Information Corporation: Satellite News, 1.
13. Allahbakhsh, M., Benatallah, B., Ignjatović, A., Motahari Nezhad, H.R., Bertino, E., & Dustdar, S. (2013). Quality Control in Crowdsourcing Systems: Issues and Directions. Internet Computing, IEEE, 17, 76-81.
14. Basili, V.R. (1996). The role of experimentation in software engineering: Past, current, and future. 18th International Conference on Software Engineering, 442-449.
15. Cavano, J., & McCall, J. (1978). A framework for the measurement of software quality. ACM SIGSOFT Software Engineering Notes, 3, 133-139.
16. Gotel, O.C.Z, & Finkelstein, A. (1994). An Analysis of the Requirements Traceability Problem. Proceedings of 1st International Conference on Requirements Engineering. IEEE Computer Society Press, 94-101.

17. Pandey, D., Khan, M.W., & Pandey, V. (2012). Role of Requirement Validation in Requirement Development. International Conference on Recent Development in Engineering and Technology, 97-100.
18. Ooi, S.M., Lim, R., & Lim, C. (2014). An integrated system for end-to-end traceability and requirements test coverage. Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, 45-48.
19. Maguire, M., & Bevan, N. (2002). User Requirements Analysis: A Review of Supporting Methods. Proceedings of IFIP 17th World Computer Congress, 133-148.
20. Peach, B., Dutoit, A.H., Kerkow, D. & Knethen, A.V. (2002). Functional requirements, non-functional requirements, and architecture should not be separated - A position paper. Proceedings of the International Workshop on Requirements Engineering: Foundations for Software Quality, 1-6.
21. Peach, B., Knethen, A.V., Doerr, J., Bayer, J., Kerkow, D., Kolb, R., Trendowicz, A., Punter, T. & Dutoit, A. (2003, May 3). An Experience-Based Approach for Integrating Architecture and Requirements Engineering. Proceedings of the 2nd International Software Requirements to Architectures Workshop, 142-148.
22. Ejaz, R., Nazmeen, M. & Zaffar, M. (2010, October 04). A quality assurance model for the analysis phase. Proceeding NSEC '10 Proceedings of the 2010 National Software Engineering Conference, 1-4.
23. Chung, L., Nixon, B.A., Yu, E. & Mylopoulos, J. (2000). Non-Functional Requirements in Software Engineering. IEEE Conceptual Modeling: Foundations and Applications, 153-160.
24. Nuseibeh, B. & Easterbrook, S. (2000, June 04). Requirements engineering: a roadmap. Proceedings of the Conference on The Future of Software Engineering, 35-46.
25. Wiegers, K.E. & Beatty, J. (2013). Developer Best Practices: Software Requirements. Microsoft Press, 68-76
26. James Gleick. (1996, December 1). A Bug and a crash. Published in New York Times magazine. Retrieved September 04, 2019, from https://around.com/ariane.html.
27. European Space Agency. (1996, June 6). Flight 501 failure- first information [Press Release]. Retrieved September 04, 2019, from http://www.esa.int/Newsroom/Press_Releases/Flight_501_failure-_first_information.