

FLOW MANAGEMENT FOR SOFTWARE DEVELOPERS IN THE KNOWLEDGE BASED ORGANIZATION FROM THE AUTOMOTIVE INDUSTRY

Adrian Bogorin-Predescu¹[0009-0000-5947-5041], Aurel Mihail Țîtu²[0000-0002-0054-6535],
Mădălina-Maria Pană³[0009-0009-8689-4343]

¹National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Splaiul Independenței no. 313, 6th District, Bucharest, Romania,

²Lucian Blaga University of Sibiu, 10 Victoriei Street, Sibiu, România,

³National University of Science and Technology POLITEHNICA Bucharest, Faculty of Industrial Engineering and Robotics, Splaiul Independenței no. 313, 6th District, Bucharest, Romania,

Emails: adrian.bogorin@yahoo.com, mihail.titu@ulbsibiu.ro, cismaru.madalina@yahoo.com

Abstract - This paper analyses the workflow of software developers in an automotive industry organization, emphasizing the crucial processes and practices that are essential for the efficiency and quality of software development. A process map was conceptualized, examined, and subsequently debated within an organization operating in the automotive industry. The software development processes of this organization have been subject to process and inspection standards reports. Examining and evaluating technological tools utilized in this procedure reveals comprehensive solutions and emerging technologies that can enhance the effectiveness of software development. Furthermore, there is a strong focus on fostering collaboration among the teams within the organization and ensuring that the workflow is flexible enough to accommodate swift changes in the industry. Automotive organizations can enhance their competitive edge by comprehending and optimizing this workflow. An appropriately organized workflow can undoubtedly result in increased efficiency and effectiveness. This aspect can result in positive results regarding the quality management of processes, with all management activities and technical tasks functioning correctly. This paper presents a detailed documentation of the workflow employed by software developers in the research and development division of the automotive industry. This aspect can lead to favourable outcomes in terms of process quality management, with all management activities and technical aspects operating accurately. The aforementioned factors have resulted in a scientific investigation that primarily relies on meticulous documentation and makes a significant qualitative contribution to enhancing the effectiveness of software development and management within an automotive organization.

Keywords: Automotive industry, Workflow, Software development, Software development, Task, Integrated system.

1. Introduction

This paper outlines the workflow of embedded software developer's workflow in the Research and Development department of an automotive organization.

Extensive research has been conducted about software lifecycles for business and commercial software, resulting in the development of numerous models tailored to various project types. A software development lifecycle model breaks down the process of software development into separate stages, with each stage having its own set of requirements, specifications, and methods. There are multiple factors that make it desirable to divide something into different stages. For instance, each

phase has the ability to manage its own quality, leading to an overall improvement in the quality of the software. In the context of larger projects, the implementation of phases can serve to delineate the responsibilities of developers and enhance the transparency of the development process [1].

The automotive industry faces mounting demands to enhance the quality and efficiency of vehicle software development. The automotive industry is subject to specific requirements and limitations, including safety, cybersecurity, and adherence to standards such as ASPICE (Automotive Software Process Improvement and Capability dEtermination) and V-model. The ASPICE standard is a prevalent framework utilized to assess and enhance the calibre of automotive software

development procedures. The V-model is a prevalent software development model that outlines the different stages of a project's lifecycle and their interdependencies [2].

Since the establishment of software engineering as a field, there has been significant focus on the initial development of dependable software within budget and according to predictable schedules. Software project managers embraced the waterfall model, the earliest process model, as it provided a way to enhance the visibility and auditability of the initial development process by using identifiable deliverables [3].

2. Management Models used in Automotive Software

Dr. Winston W. Royce is generally credited with introducing the Waterfall model in management, specifically in the field of software development.

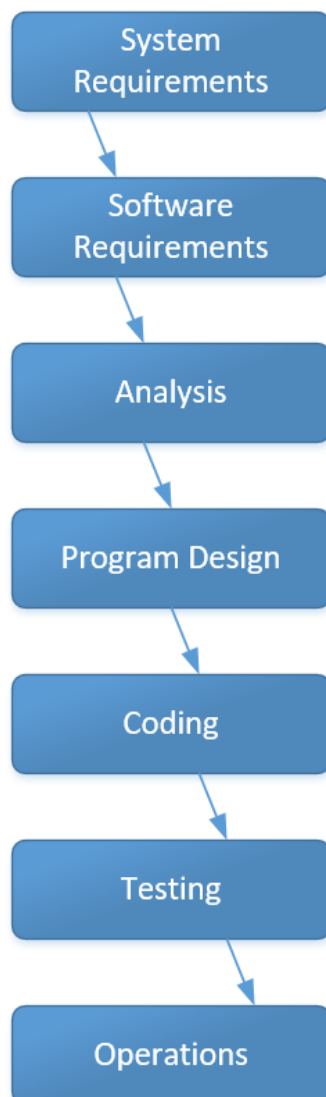


Figure 1: Waterfall model concept

Despite its widespread adoption and adaptation, the Waterfall model has faced criticism for its

Royce presented the model in his 1970 publication entitled "Managing the Development of Large Software Systems" [4]. Royce's paper outlined a sequential design methodology for software development, later named the Waterfall model due to its linear and cascading nature, where each phase seamlessly transitions into the next.

The model, from Figure 1, follows a structured approach where the development process is divided into separate phases. These phases include requirements analysis, system design, implementation, testing, deployment, and maintenance. Thorough documentation and planning are emphasized at each stage, as each phase must be finished before the next one starts.

Despite its widespread adoption and adaptation, the Waterfall model has faced criticism for its inflexibility and challenges in accommodating changes once the process has begun. As a result, there was a rise in the adoption of iterative and agile methodologies in software development.

The waterfall has two modified versions: the V-shaped model and the incremental technique. The V-shaped model distinguishes itself by conducting testing after each stage, which significantly improves the product's final quality. While the stages of the incremental model occur gradually. However, each stage occurs multiple times throughout the software's life cycle. The process of developing a program with numerous planned functions starts with the implementation of the basic version [5].

The V-model is a systematic approach applied to complex projects, particularly those of significant scale, where comprehensive test strategies are developed for each stage of testing subsequent to the implementation of the requirements and functions. The V-model is highly aligned with regulatory standards as it enables organizations to produce the required outputs in order to comply with regulatory requirements. The development and testing phases are clearly defined, ensuring that there is no remaining testing work as validation is conducted at each stage of sustainable development. As a result, the overall number of faults will be reduced [6].

The V-model, depicted in figure 2, is predominantly utilized in the automotive sector, particularly within the realm of research and development. It encompasses both the domains of mechanics and electronics, with a primary focus on software-related tasks [7].

The authors in [8] provide a detailed description of the application of the PLC (Product Life Cycle) in the automotive industry. This includes all stages starting from the conceptualization, development, production, and concluding with the final phase of product recycling.

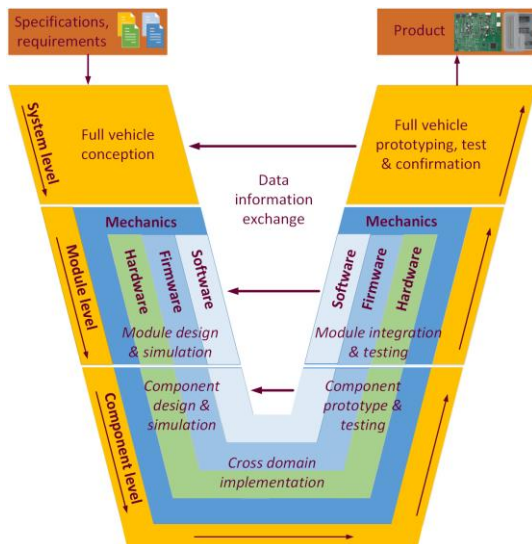


Figure 2: V-model for automotive development

The complexity of automotive software systems necessitates a systematic approach to managing software product lines. Knieke et al. emphasize the importance of structured methodologies for the evolution of automotive software architectures, which can help streamline development processes and facilitate knowledge sharing among teams [9]. This structured approach is particularly relevant in the context of integrating various software components, where effective flow management can mitigate risks associated with ambiguous requirements and integration challenges [10]. The need for a cohesive framework that encompasses both agile practices and structured methodologies is evident in the ongoing efforts to enhance collaboration across teams and improve the overall software development lifecycle.

In addition to agile methodologies and structured frameworks, the role of advanced testing methodologies cannot be overlooked.

Hodel et al. propose a systematic methodology for functional testing of automotive embedded software, which is essential for ensuring that the software meets safety and performance criteria [11]. This approach not only enhances the robustness of the software but also contributes to effective flow management by identifying potential issues early in the development process. Furthermore, the integration of fault injection testing, as discussed by Park and Choi, is crucial for validating the functional safety of automotive software, thereby ensuring that systems can operate safely even in the presence of faults [12].

3. Software Developer's Workflow

Typically, automotive industry organizations in Engineering Centres adopt software platforms to manage the workflow of software activities and the phases of PLC management phases. One such platform is the PTC (Parametric Technology Corporation) Integrity Lifecycle Manager. The PTC software operates on a client/server architecture, offering both desktop application and web client graphical user interfaces.

The software offers a collaborative platform for software development organizations to oversee the entire development process, including requirements management, engineering change management, revision control, build management, test management, software deployment, and generating relevant reports and metrics.

The Integrity platform operates on the Waterfall methodology and functions at the task level. Typically, SW development engineers receive these tasks from PMs (Project Managers), test engineers, or even other SW engineers. Tasks are categorized into various groups: change (change request), implementation, and software repair or bug fix.

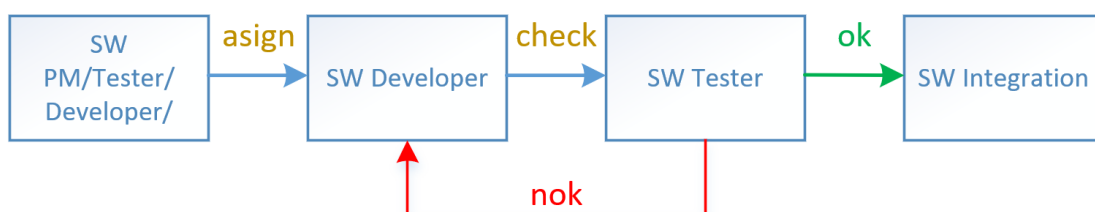


Figure 3: Process order for a SW implementation/change

Figure 3 shows the sequential steps involved in implementing a software change. The sequential progression of a software change involves the task initiator (who may be a project manager, tester, or software developer) passing the change to the software developer. Subsequently, the change is subjected to testing by the tester, and ultimately, it gets integrated into the final software package.

Upon identifying an error in the Software, the project manager assigns an analysis task to the

software developer. Once the software developer analyses the problem, it identifies the underlying causes that resulted in this software error. Subsequently, they implement the necessary modifications to rectify the error in the software. Once the developer has implemented the software fix, it is then verified by the test engineer. If additional anomalies are identified in the software during the testing process, or if any non-compliances persist, the responsibility for resolving the issue

reverts, back to the software developer. However, if the software tester discovers during the testing phase that the software has been repaired, the updated software will be incorporated into the primary software. Given the circumstances, the task can be finished.

Upon receiving a task for implementation, the assigned software developer is notified via email.

Figure 4 illustrates the various states that a task can have. The main states of the task are:

- OPEN;
- ANALYSIS;
- IN_WORK;
- IN_VERIFICATION;
- SOLUTION_PROVIDED;
- CLOSED.

Figures 6, 7, 8, and 9 present the individual state of each phase depicted in the diagram shown in figure 4.

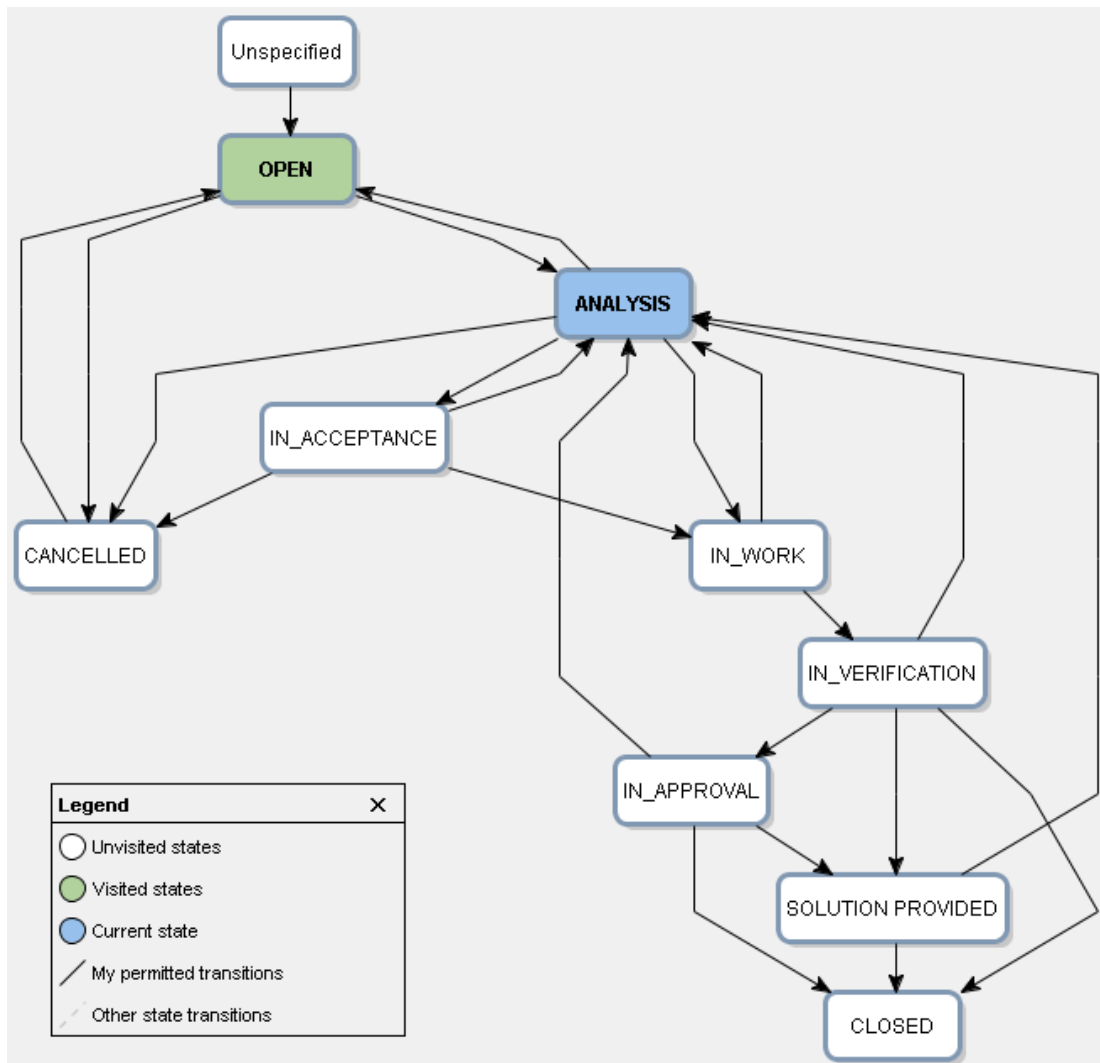


Figure 4: Software Task workflow diagram

Figure 5 displays the graphical user interface of the Integrity platform, providing the up-to-date status of the task.

The current state of a task can be evaluated in its "HEADER" segment. Currently, the task is under analysis. The "Summary" field provides a concise description of the problem, which corresponds to the task name in the Integrity platform. The "Champion" field denotes the project manager's name, who holds the responsibility for the project.

The software developer's name appears in the "Assigned User" field, while the software tester's name is inputted in the "Assigned Tester" field. In the right portion of figure 5, the deputies are typically assigned to the personnel who are directly responsible for the task, regardless of whether they are champions, software developers, or testers.

When the task is handed over to the software developer, it enters the first stage known as the "OPEN" state, as seen in figure 4's state diagram. Figure 6 depicts a task in the "OPEN" state.

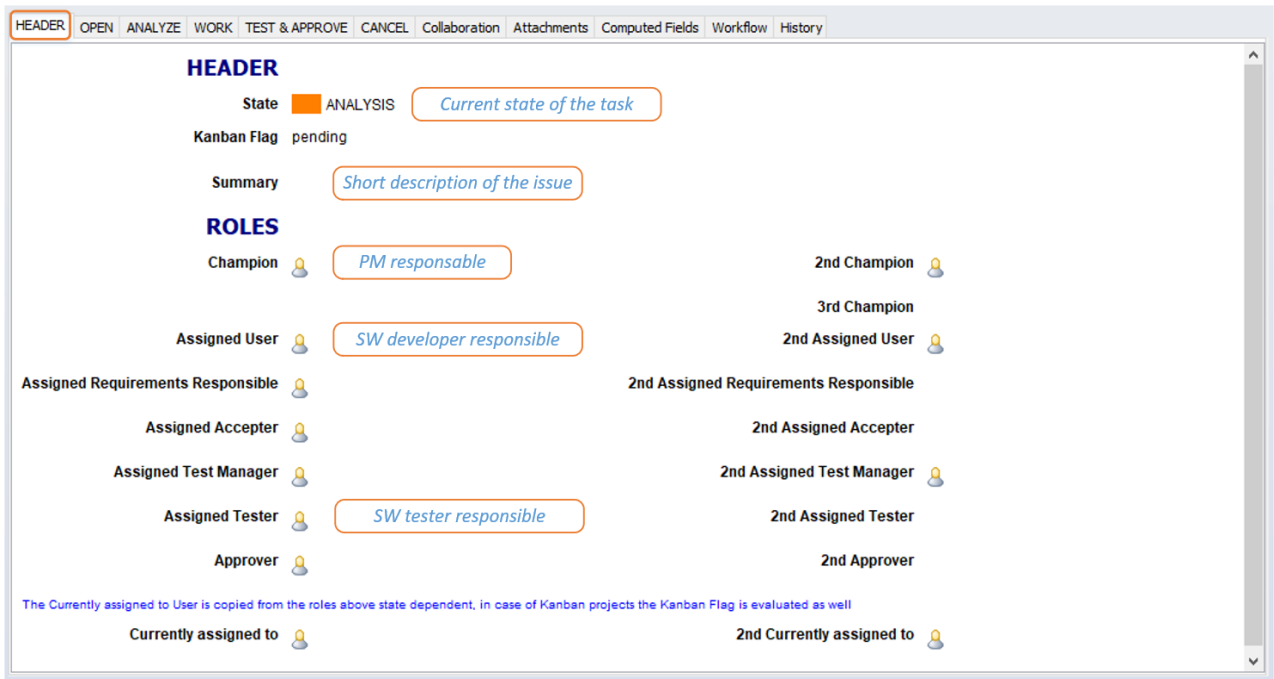


Figure 5: Integrity – Header of the Task

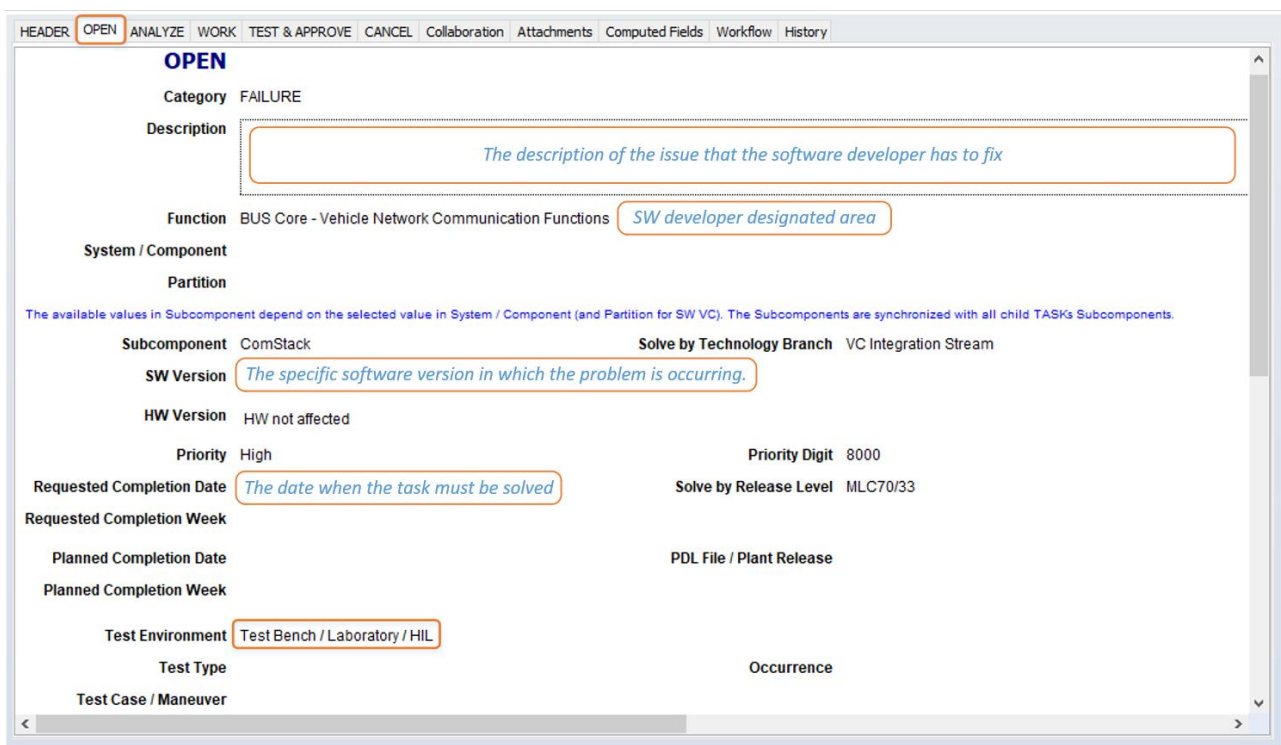


Figure 6: Integrity – Open state of the Task

The "Description" container provides an extensive description of the malfunctioning behaviour observed in the software. The "Function" field is targeted at software engineers specializing in a certain domain. Given the intricate nature of automotive software designed for Electronic Control Units (ECUs) used in automobiles, with around 1 million lines of code, the software is organized into separate modules based on their specific functions.

The software is partitioned into several intricate domains in this context:

- "Vehicle Network Communication Functions" with the "ComStack" subcomponent, is responsible for the communication between the ECU and the other ECUs in the car;
- "Operation System" is responsible for the operating system incorporated in the ECU;

- "Diagnosis" is responsible for the diagnostic part of the ECU. It is the area where the ECU error codes are stored.

The specific software version that indicates the corresponding issue is listed in the "SW version" container. The term "Requested Completion Date" refers to the specific date by which the software issue is expected to be resolved. The significance of this date is contingent upon the gravity of the issue and the level of advancement of the project. If the software has already been integrated into the vehicles now operating on the streets, then the date becomes very crucial and must be rigorously adhered to. The "Test Environment" field specifies the location where the engineer in charge of testing will evaluate the program.

The test may be conducted in three locations: the work bench where the ECU is mounted, the test laboratory, and the HIL (Hardware in the Loop) system, which is an advanced system that simulates the car's environment.

Figure 7 depicts the state of the work when the software developer examines a bug in the program.

The software developer has to indicate in the "Estimated Effort" section how long he believes it will take him to identify the issue's root cause. The duration of this interval period might vary from few hours to several days, depending on the complexity of the situation. Oftentimes, the programmer's time estimate is undervalued, which may result in significant delays in software repairs.

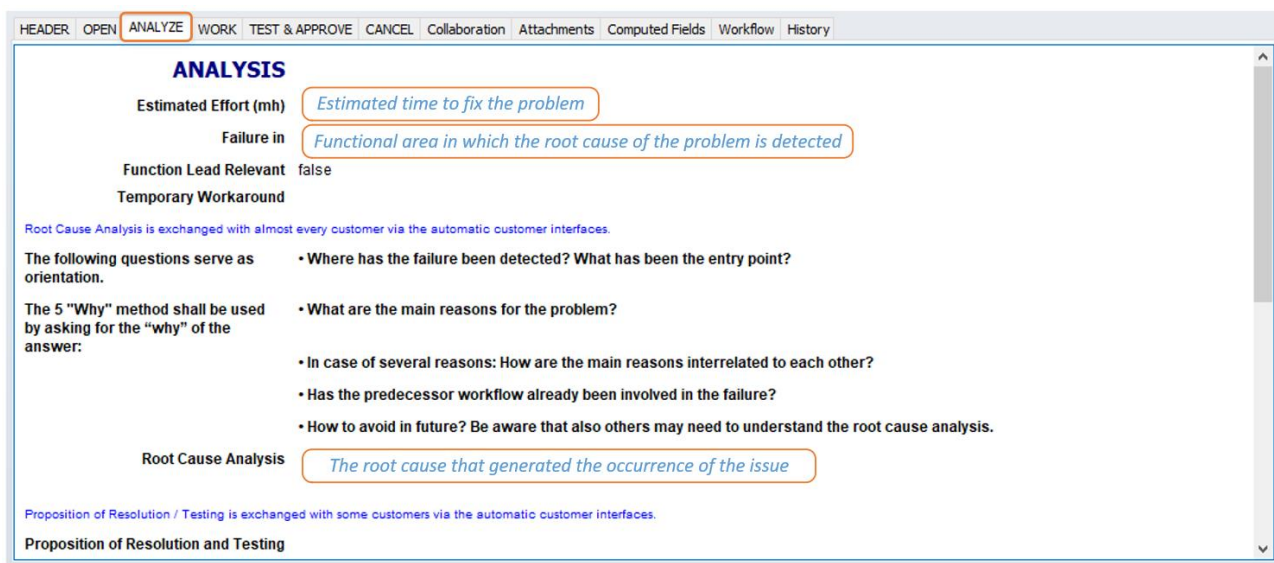


Figure 7: Integrity – Analyse state of the Task

Once the software developer determines the root cause of the issue, they proceed to choose the functional area where the cause was found into the "Failure in" box. Frequently, there is a discrepancy between this region and the "Function" category in the "Open" status of the task in question. This might be credited to the complexities of the project or the inadequate expertise of the project manager or the

person responsible for assigning the task to the programmer.

The software developer enters the reason that caused the program mistake into the "Root Cause Analysis" area.

Upon completion of the analysis, it will be certain which modifications are necessary to resolve the software issue.

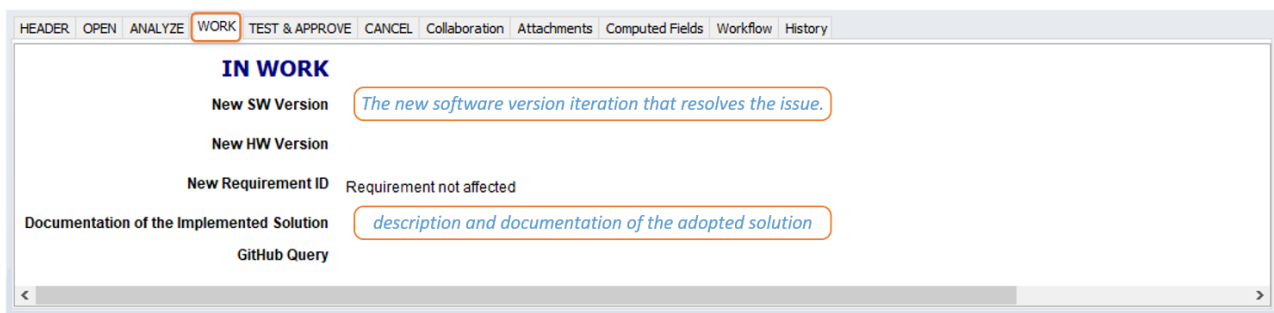


Figure 8: Integrity – Work state of the Task

The task transitions from the "ANALYZE" state to the "WORK" state, as illustrated in figure 8, and the

software developer implements the modifications that resolve the software issue. The "New SW

Version" field is inputted with the number of the new software version that contains the update. In case that the requirements require modification following the software implementation, the "New Requirement ID" element is used to indicate this. The documentation that supports the software change and the software developer's method for implementing it is included in the "Documentation of the Implemented Solution" container.

The test engineer receives email notification that he has software to verify after the software developer moves the task from the "WORK" to the "IN_VERIFICATION" state.

The task is shown in Figure 9 as "TEST&APPROVE" state. Software on the ECU is first checked by the software test engineer.

After testing the new SW version, in the "TEST & APPROVE" section, enter the test verdict in the "Test Evaluation" field: PASS or FAIL. If the SW has not passed the tests, then it is the responsibility of the SW Tester to return the task to the ANALYSIS state. In this way the SW developer is notified by email and on the Integrity platform that his implementation did not give the desired result and he must rethink the SW modification. When the SW passes the tests, the software test engineer marks the task as "SOLUTION PROVIDED" and enters the PASS verdict in the "Test Evaluation" section. The SW developer then marks the task as "CLOSED". The test engineer has to submit the test report in the "Documentation / Link of Test Result" section in either case PASS or FAIL.

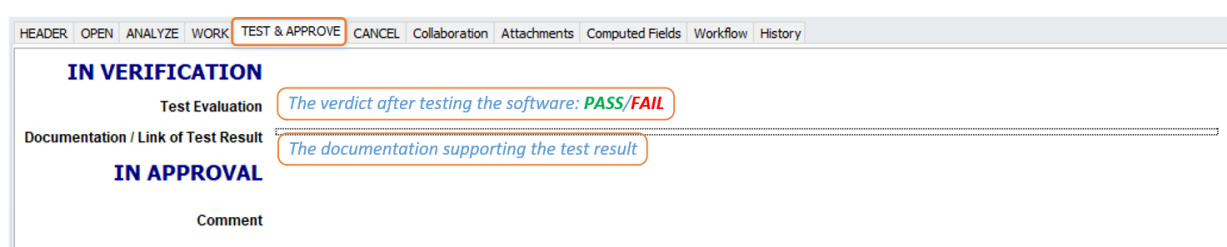


Figure 9: Integrity – Test and approve state of the Task

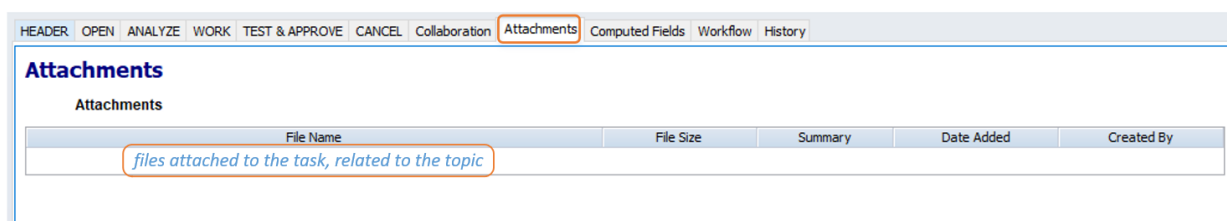


Figure 10: Integrity – Files attached to the Task

The Integrity platform provides the capability to associate documents with a task, as seen in figure 10.

In some scenarios, it is necessary to have items connected to the work, such as documentation, log files generated by the diagnostic process in the vehicle, and screenshots. These resources assist the software developer in analysing what is at issue.

4. Conclusions

The integrated Integrity platform enables traceability between processes. The Integrity platform facilitates seamless communication between the software development processes, testing, and integration into the final program. Furthermore, globalization has facilitated the execution of tasks in many places worldwide via the use of the Internet and Virtual Private Network (VPN). Process management systems facilitate the progression of Product Life Cycle stages. Until a decade ago, the product development cycle in the automobile industry typically spanned around four years. The duration of the development process, from the first concept design to the final series

manufacturing, was reduced by half, resulting in a timeframe of 2 years. One of the factors contributing to this issue is the intense rivalry among TIER 1 suppliers in the automobile industry. The supplier who is able to provide the product in a shorter amount of time will be the one who gets the project. Certainly, this is also evident in the quality and development of the product, which, although being completed in terms of mechanics and electronics, still has a long way to go in terms of software development. Ongoing progress in development and repair of software defects, especially at the production stage. An inherent benefit of the software is its seamless upgradability post-installation on the vehicle.

References

[1] Dubey, A., & McInnes, L. C. (2017). Proposal for a Scientific Software Lifecycle Model. In Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational and Data-enabled Science & Engineering. SC '17: The

- International Conference for High Performance Computing, Networking, Storage and Analysis. ACM. <https://doi.org/10.1145/3144763.3144767>
- [2] Kronic, M. V. (2023). Documentation as Code in Automotive System/Software Engineering. In *Elektronika ir Elektrotechnika* (Vol. 29, Issue 4, pp. 61–75). Kaunas University of Technology (KTU). <https://doi.org/10.5755/j02.eie.33843>
- [3] Bennett, K. H., Rajlich, V. T., & Wilde, N. (2002). Software Evolution and the Staged Model of the Software Lifecycle. In *Advances in Computers* (pp. 1–54). Elsevier. [https://doi.org/10.1016/s0065-2458\(02\)80003-1](https://doi.org/10.1016/s0065-2458(02)80003-1)
- [4] Royce, W. W., (1970), Managing the Development of Large Software Systems. *Proceedings of IEEE WESCON*, 26, 328-388
- [5] Rozhnova, T., Tomachynska, V., & Korsun, D. (2022). Life cycle models, principles and methodologies of software development. In *InterConf* (Issue 28(137), pp. 394–401). Scientific Publishing Center InterConf. <https://doi.org/10.51582/interconf.19-20.12.2022.040>
- [6] Ghnaimat, T. M., & Hudaib, A. (2022). Hybrid software process model: V-SCRUM. In 2022 International Conference on Emerging Trends in Computing and Engineering Applications (ETCEA). 2022 International Conference on Emerging Trends in Computing and Engineering Applications (ETCEA). IEEE. <https://doi.org/10.1109/etcea57049.2022.10009846>
- [7] Hirz, M., (2018). An approach supporting integrated modeling and design of complex mechatronics products by the example of automotive applications, Conference: 22nd Multi-Conference on Systemics, Cybernetics and Informatics – WMSCI
- [8] Bogorin-Predescu, A., Țîțu, S., & Țîțu, A. M. (2023). Product Life Cycle in Automotive Industry. In *Lecture Notes in Networks and Systems* (pp. 411–417). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-31066-9_45
- [9] Knieke, C., Rausch, A., Schindler, M., Strasser, A., & Vogel, M. (2022). Managed Evolution of Automotive Software Product Line Architectures: A Systematic Literature Study. In *Electronics* (Vol. 11, Issue 12, p. 1860). MDPI AG. <https://doi.org/10.3390/electronics11121860>
- [10] Schroeder, J., Berger, C., & Herpel, T. (2015). Challenges from Integration Testing using Interconnected Hardware-in-the-Loop Test Rigs at an Automotive OEM. In *Proceedings of the First International Workshop on Automotive Software Architecture* (pp. 39–42). *CompArch '15: Federated Events on Component-Based Software Engineering and Software Architecture*. ACM. <https://doi.org/10.1145/2752489.2752497>
- [11] Hodel, K. N., Reinaldo Da Silva, J., Yoshioka, L. R., Justo, J. F., & Santos, M. M. D. (2022). FAT-AES: Systematic Methodology of Functional Testing for Automotive Embedded Software. In *IEEE Access* (Vol. 10, pp. 74259–74279). Institute of Electrical and Electronics Engineers (IEEE). <https://doi.org/10.1109/access.2021.3128431>
- [12] Park, J., & Choi, B. (2020). ASFIT: AUTOSAR-Based Software Fault Injection Test for Vehicles. In *Electronics* (Vol. 9, Issue 5, p. 850). MDPI AG. <https://doi.org/10.3390/electronics9050850>