UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN ICT FOR INTERNET AND MULTIMEDIA

"Automated environment for the analysis of ASPICE compliance for automotive FW"

Relatore:
Prof. GIAN ANTONIO SUSTO

Correlatore:
Ing. FRANCESCO SARTORIO
Ing. DENIS DONI

Laureanda:
NORHEN ABDENNADHER

ANNO ACCADEMICO: 2023 – 2024

Data di laurea: 03/07/2023

Abstract

This thesis summarizes the work carried out as part of the end-of-studies project to obtain a master's degree from the University of Padova, performed as an internship in collaboration with Infineon Technologies and the University of Padova.

Automotive Software Process Improvement and Capability Determination is the abbreviation for Automotive Spice (ASPICE). It was developed to evaluate the effectiveness of original equipment manufacturer (OEM) suppliers' product development processes in the automobile industry. To guarantee the greatest level of embedded automotive software development quality, it outlines best practices and procedures. The audit performed by outside, impartial, and ASPICE-certified assessors serves as the foundation for the certification procedure.
In this thesis, we define and implement an automation environment that allows the analysis of work products related to Firmware development in order to evaluate the state of ASPICE compliance. The dashboard introduces a tool to fetch, store, and visualize the data related to different software programs, as well as the result of each undergoing test. Python scripts were set in place to do the extraction of the data as well as populate the tool's data in a MySQL database. Finally, a Tableau dashboard was implemented and integrated to display this data in a clear and approachable way, that facilitates the decision-making process related to the ASPICE compliance state.

Keywords: ASPICE, Automotive SW/FW, Quality standards evolution, Key performance indicator (KPI), V-model, Continuous Integration, Data fetching, Database management systems, Web scrapping, Application programming interface, and Project management.

Dedications

“

To my family, thank you for all your support, help, prayers, and advice, thank you for believing in me, for being strong and patient, for appreciating my achievements, and finally, thank you for what you taught me to be the person I am today. This work is the culmination of your great efforts. May God Bless You with Good Health and Long Life,

To my parents, Tarek and Sihem, so many words cannot express my love and affection. You have always supported and encouraged me throughout my journey. On this memorable day, for me as well as for you, receive this work as a sign of my deep appreciation and my profound esteem,

To my brother and sister, Riadh and Rihab, I can never imagine how the world would be without you. Both of you have been my best cheerleaders. Wishing you all luck and success in your lives,

To someone close to my heart and far from the eyes, someone who was there for me in my ups and downs. To my beloved Neji, the one I always rely on when I felt anxious and stressed, I cannot express my deepest gratitude for the continuous support you have always shown. Thank you for everything you have added to my life.

To all my dear ones, to all of you

Thank you.

”

- Norhen

Acknowledgement

I have been lucky enough to have a chance to spend a few months working at Infineon Technologies Padova, Italy. A rich and highly dynamic learning environment provided me with valuable hands-on experience and a solid knowledge base for my current and future work.

I would like to take this opportunity to express my heartfelt gratitude to a number of people whose I was blessed to know and work with. My 20 years of studies had come to a successful conclusion thanks to you.

First, to my thesis tutor at the University of Padova **Prof. Gian Antonio Susto** for his valuable support and advice during the period of the internship.

To my thesis supervisor at Infineon company **Mr. Doni Denis**, the one I turn to whenever I run into a trouble spot, was an incredible mentor to me. I can't forget his generosity and his kind support. He was always there to provide me with valuable suggestions to make my internship succeed and faces all the encountered problems. Thank you for all the lessons you've given me, and for trusting my skills. Without your guidance, I wouldn't make it this far.

To my thesis tutor at Infineon Technologies **Mr. Francesco Sartorio** I would like to express my heartfelt gratitude for his unwavering support and encouragement throughout my internship. He has created an exceptional work environment to foster my learning. I am thankful for his belief in my abilities and for entrusting me with valuable opportunities to explore the professional world.

Thanks to all of the UNIPD teachers who were involved in my education during my time at the university, and to all Infineon colleagues who have never hesitated to provide help. I am fortunate to be part of such a collaborative and supportive team that has embraced a culture of mutual growth and development.

I would like to express my gratitude to the members of the jury for the honor they have bestowed upon me by reviewing my graduation project, and I hope that they can find the clarification and inspiration that they seek in this report.

Contents

List of Figures

List of Tables

List of acronyms

ASPICE	*Automotive Software Process Improvement and Capability dEtermination*
SW	*SoftWare*
FW	*FirmWare*
OEM	*Original Equipment Manufacturer*
KPI	*Key Performance Indicator*
CI	*Continuous Integration*
DBMS	*DataBase Management System*
API	*Application Programming Interface*
BP	*Body Power*
SQL	*Structured Query Language*
CR	*Change Request*
PR	*Problem Report*
CCB	*Control Change Board*
SWRQ	*Software ReQuirements*
SWA	*Software Architecture*
SWUD	*Software unit design*
SWDI	*Software Design Implementation*
CC	*Cyclomatic Complexity*

MISRA	*Motor Industry Software Reliability Association*
MC/DC	*Modified Condition/Decision Coverage*
ITCov	*Integration Test Specification Coverage*
UTCov	*Unit Test Specification Coverage*
SOAP	*Simple Object Access technology*
REST	*Representational state transfer*
DBMS	*Database Management System*

General Introduction

In today's world, embedded systems have become ubiquitous, powering everything from mobile devices to industrial machinery. In particular, firmware plays a critical role in enabling the functionality of these embedded systems, providing the low-level software necessary to interface with hardware components and execute system-level tasks. In the automotive industry, firmware is essential to the operation of vehicles, controlling everything from engine timing to infotainment systems. As vehicles have become an essential part of our transportation system, and with the increasing demand for more advanced and efficient solutions, the automotive industry is constantly evolving [1].

As automotive systems have become more complex and interconnected, ensuring the quality and reliability of firmware has become increasingly important, and the need for sophisticated software to control all operations has increased. Firmware plays a critical role in ensuring that the car operates smoothly, efficiently, and safely, and this complexity brings with it many challenges including the need for rigorous testing and validation [2]. One approach to achieving this, is through compliance with the Automotive SPICE (ASPICE) standard. ASPICE [3] is a framework for assessing and improving the software development processes used in the automotive industry by providing guidelines for the development process, ensuring that the software meets the necessary quality and safety requirements. By adhering to ASPICE, automotive companies can ensure that their firmware development processes are well-defined, well-controlled, and capable of delivering high-quality and reliable firmware.

The implementation of ASPICE in automotive firmware development is critical. It's quite a challenge to ensure that the software meets the necessary safety standards and that the car operates as intended. Without proper implementation of ASPICE, the risk of software errors, bugs, and vulnerabilities increases, potentially leading to safety hazards and costly recalls.

Therefore, in this thesis, we will discuss the importance of implementing ASPICE in automotive firmware development. We will examine the various aspects of ASPICE, including its benefits, challenges, and how it can be integrated into the software development process. Additionally, we will automate an environment that allows the analysis of work products related to the firmware development in order to evaluate the state of ASPICE compliance.

This report provides a comprehensive overview of my master's degree graduation project conducted at the University of Padova. It is structured into five chapters, starting with an introduction to the project's context and scope. Then we discussed the significance of ASPICE in evaluating software quality in the automotive industry. The subsequent chapter focuses on key performance indicators (KPIs) and the various techniques employed to retrieve data for the ASPICE dashboard. The database design and the seamless integration of the dashboard are presented in detail. Finally, the report concludes with a summary of the project and outlines potential avenues for future research and development.

Chapter 1

Internship context & Scope Definition

Introduction

This report is the result of a master thesis stage which is realized during a six months internship in Infineon's Development Center at Padova. As a member of the body power team, I had the chance to work on the subject given to me and to present a solution that will be implemented to achieve the desired goals, and this report comes as a summary of every step of the thesis project. This first chapter aims to introduce the surroundings of this thesis as well as some technical elements for the sake of setting the rest of the report clear and understandable. Finally, the problem statement and the thesis goals will be presented at the end of this chapter.

1.1 The host enterprise

In this section, we will present the host enterprise and its activity sector.

1.1.1 Infineon Technologies

Infineon Technologies [4] is a leader in the world of semiconductor solutions that make life easier, safer, and greener. It is a leading innovator in the international semiconductor industry founded in April 1999. It is headquartered in Munich, Germany but, it has 17 Production sites and 35 Research and Development ones scattered all over Europe, the Americas, and the Pacific Regions, and more than 40k employees worldwide.

Figure 1.1: Infineon's Logo

Semiconductors, which are hardly visible, have evolved into a need for daily living. With microelectronics that connect the physical and digital worlds, Infineon is instrumental in defining a brighter future. Their semiconductors make it possible to manage energy effectively, move about intelligently, and communicate securely and easily in a linked environment.

1.1.2 Sector of activity

Today the company is working on 4 business areas. Here follows the official presentation given by the company for these 4 areas :

- Automotive (ATV): In the ATV segment, Infineon actively participates in defining the industry's fundamental trends as well as creating products and solutions for traditional drive trains. The expanding number of electronic applications in automobiles, which is a trend further amplified by the rising popularity of electro-mobility, is driving up demand for our power semiconductors. We are the undisputed market leader in silicon-based IGBTs and IGBT modules. Our expertise in silicon carbide is also increasingly relevant for automotive power semiconductors. We are paving the way for self-driving cars with our radar sensors and microcontrollers. Positioned as number two in the radar sensor market, we are already noting strong momentum from the proliferation of driver assistance systems. In the long term, radar systems will be fused with other sensor technologies. We are laying the groundwork for this by developing products such as LIDAR solutions. With our AURIX™ family, we are also benefiting from the trend toward increased automation. Our products here control electronic systems such as steering and braking, also acting as host controllers to provide functional safety and data security for central computing platforms.

- Industrial Power Control (IPC): This industry sector focuses on the effective conversion of electric energy across the whole supply chain, from generation and transmission all the way through to consumption. Here, applications include electric car charging stations, home appliances, high-voltage DC transmission systems, energy storage systems, and wind turbines. The market leader for power semiconductor modules and discrete power semiconductors based on IGBTs is Infineon. We intend to achieve technological leadership in silicon carbide in order to further enhance this core IPC business. We are also placing more emphasis on complementary product categories, particularly Intelligent Power Modules (IPMs), which integrate controllers, drivers, and switches to allow digital control capabilities.

- Power Management and Multimarket (PMM): Our PMM section concentrates on power semiconductors for energy management as well as parts for mobile devices and wireless networks. For applications in sectors like aerospace, PMM also specializes in incredibly dependable parts. Infineon is without a doubt the market leader for MOSFETs worldwide. Excellent levels of energy efficiency are provided by our CoolMOSTM and OptiMOSTM series. Additionally, we provide cutting-edge technologies built on gallium nitride. We are working to broaden our selection of complimentary drivers and controllers in addition to this product category. One of the power semiconductors' most rapidly expanding uses is in battery-powered gadgets. We have a significant technological presence in the high-frequency and sensor arena thanks to our MEMS microphones (silicon in particular), time-of-flight sensors for 3D cameras, and radar applications. In each market, we have already built quite strong positions. At the same time, we can apply our knowledge in these fields to a growing number of use cases that will take off in the upcoming years. Facial recognition and human-machine interaction (HMI) are two important examples here.

- Digital and Security Solutions (DSS): The Digital Security Solutions (DSS) segment has over thirty years of experience delivering some of the world's most challenging and large-scale digital security projects. Our success here is built on our wealth of expertise in conventional smart card applications. We are transferring our core skills in payment cards and government documents to the fast-growing field of embedded security applications. As digitization shapes more and more areas of everyday life, security is becoming a key success factor for applications across industries as diverse as computing, automotive, Industry 4.0 and smart homes. Parallel to its role as an independent business segment, DSS acts as a competence center for our other three segments, supporting their efforts to hardware security functionality into their respective system solutions. Working in all these fields Infineon keeps tight relationships with many customers. Some of them are shown in Figure 1.2.

1.1.3 Infineon technologies, Padova Italy

Established in 1999 and operational since 2000, Infineon Technologies Italy is a fully owned subsidiary of Infineon Technologies BV and with offices in Milan for Central Functions and Sales, Padova and Pavia for R&D activities. The Development Centre Pauda was founded in 2001 and focuses its activities on:

- Competence center for DCDC, NVMs and Functional Safety
- Product Development of Power Management ICs
- Technical Marketing and Product Development of LED lighting solutions.
- Technical Marketing, Application and Product Engineering of Micro-controllers Non-Volatile Memories

1.2 Thesis overview and problem statement

In this section, we will provide the motivation and objectives of our work. We will also provide a general overview to figure out the current problem and the motivation to implement an automated ASPICE environment.

1.2.1 Problem Statement & Motivation

Everything from mobile gadgets to industrial machines is now powered by embedded systems. Firmware, in particular, is crucial in allowing the functionality of these embedded systems because it offers the low-level software required to communicate with hardware parts and carry out system-level operations. Firmware is crucial to the running of automobiles in the automotive industry, handling everything from engine timing to entertainment systems.

Over the previous few decades, vehicle software has expanded tremendously, moving from having zero lines of code to as much as 200 million in certain situations today. Projects may require the cooperation of hundreds of engineers to satisfy tens of thousands of criteria due to their complexity. Making ensuring everyone is on the same page is crucial if OEM specifications are to be met. I'll give you an example. It is customary to do numerous walkthroughs while a house is being built by a construction company to make sure the authorized building plans are being followed. When a significant mistake, such as bad plumbing, is discovered early on in the project, it is easier and less expensive to fix. The same is true for software development, however, it might be more difficult to see clearly through a software project while it is being created. In reality, OEMs may feel safe knowing that ASPICE will take their projects' demands into consideration at every level because of the fact that this is where ASPICE operates.
An industry-standard method for assessing software development processes is Automotive Spice or ASPICE. Since its launch in 2005, ASPICE has assisted automotive suppliers in implementing best practices to find flaws earlier in the development process and guarantee that OEM specifications are satisfied. OEM found that the average ASPICE supplier discovered 90 percent of defects 11 months before the start of production, whereas the average non-ASPICE supplier discovered 90 percent of defects just two months before the start of production, endangering an on-time launch.

Because of this, ASPICE is seen as a crucial stage in incorporating best practices, spotting flaws earlier in the development process, and guaranteeing that OEM standards are followed. What therefore is the secret to assessing compliance and making ASPICE data-driven decisions?
Understanding data is essential in today's world if you want to run any organization successfully. Even the most data-savvy individual may find it difficult to handle the amount of information that is always at their disposal. Making data-driven ASPICE choices requires locating the most crucial facts and organizing them in a way that is simple to comprehend. The creation of a dashboard that conveniently shows all of your data visualizations in one location is one of the simplest methods to make data simple to grasp for both technical and non-technical users. Your data will become out of date if you have to manually import, export, and convert it from one platform to another before your dashboard can display it. This will result in inefficiencies at best and erroneous assumptions and serious decision-making errors at worst. For this reason, finding an automated system that collects and presents all necessary data in real-time to serve as a rapid and precise reference for its users is quite crucial.

1.2.2 Objectives & Proposed solution

The major goal of this work is to define and put into place an automated framework that enables the examination of work products for firmware development on a weekly basis in order to assess ASPICE compliance.
ASPICE data of different sorts may be graphically displayed in one location using this automated system. Typically, a dashboard's purpose is to present various, linked facts in an easy-to-understand style. The organization has to be able to quickly observe and comprehend things like key performance indicators (KPIs) or other crucial business in-

formation.

The goal is to develop a dashboard that provides each project with real-time ASPICE-related data updates because, in today's business, data can change within hours or even minutes. Stakeholders can easily keep a close check on activities that may be going behind or projects that are underperforming with the use of a dashboard, which acts as a fast reference point for them so they can take remedial action.

1.2.3 Project Workflow

The workflow of the project is divided into four main parts:

- Fetching Data from Different Tools: In this step, data is collected from various tools and sources relevant to the project. These tools may include version control systems, bug tracking systems, testing frameworks, and other software development tools. The data collected includes information about code changes, bug reports, test results, and other relevant metrics.

- Storing Data into a Database: Once the data is fetched, it needs to be stored in a MySQL database for further processing and analysis. The data is organized and structured in a way that facilitates easy retrieval and manipulation for subsequent steps.

- Integration with Jenkins for Automation: After the data is stored, the next step is to integrate the scripts and processes with Jenkins, a popular automation server. Jenkins allows for the automation of various tasks, such as running tests, generating reports, and performing continuous integration and deployment. By integrating the scripts with Jenkins, the project can benefit from automated workflows and streamlined processes.

- Visualizing Data with Tableau: The final step involves visualizing the collected and processed data using Tableau, a powerful data visualization tool. Tableau provides a range of features and visualizations that allow for interactive exploration and analysis of the data. By visualizing the data, project stakeholders can gain valuable insights, identify patterns, and make data-driven decisions.

By following this workflow (Check Figure 1.2), the project can effectively gather data from different tools, store it in a database, automate processes using Jenkins, and visualize the data using Tableau, enabling efficient analysis and informed decision-making.

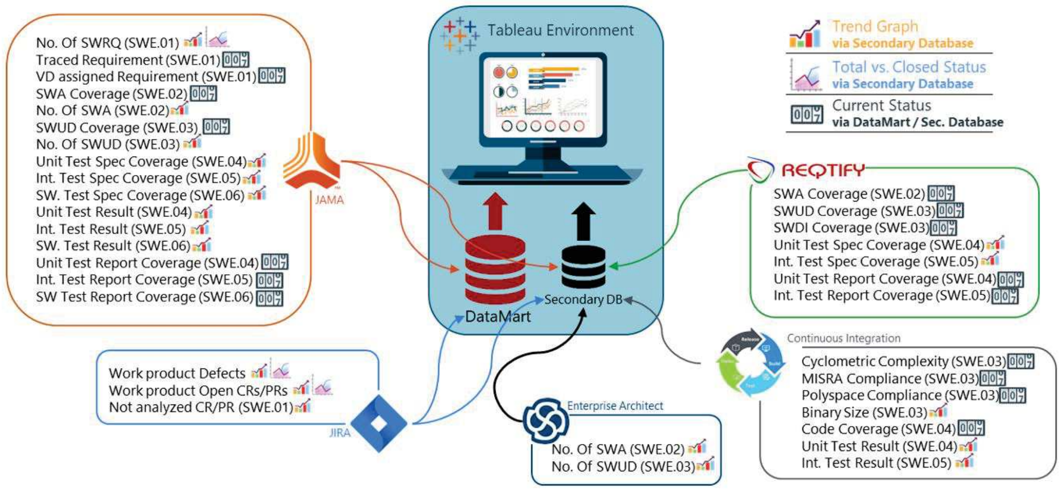


Tableau Environment
Trend Graph
via Secondary Database
Total vs. Closed Status
via Secondary Database
Current Status
via DataMart / Sec. Database
No. Of SWRQ (SWE.01)
Traced Requirement (SWE.01)
VD assigned Requirement (SWE.01)
SWA Coverage (SWE.02)
No. Of SWA (SWE.02)
SWUD Coverage (SWE.03)
No. Of SWUD (SWE.03)
Unit Test Spec Coverage (SWE.04)
Int. Test Spec Coverage (SWE.05)
SW. Test Spec Coverage (SWE.06)
Unit Test Result (SWE.04)
Int. Test Result (SWE.05)
SW. Test Result (SWE.06)
Unit Test Report Coverage (SWE.04)
Int. Test Report Coverage (SWE.05)
SW Test Report Coverage (SWE.06)
JAMA
DataMart
Secondary DB
REQTIFY
SWA Coverage (SWE.02)
SWUD Coverage (SWE.03)
SWDI Coverage (SWE.03)
Unit Test Spec Coverage (SWE.04)
Int. Test Spec Coverage (SWE.05)
Unit Test Report Coverage (SWE.04)
Int. Test Report Coverage (SWE.05)
Work product Defects
Work product Open CRs/PRs
Not analyzed CR/PR (SWE.01)
JIRA
Enterprise Architect
No. Of SWA (SWE.02)
No. Of SWUD (SWE.03)
Continuous Integration
Cyclometric Complexity (SWE.03)
MISRA Compliance (SWE.03)
Polyspace Compliance (SWE.03)
Binary Size (SWE.03)
Code Coverage (SWE.04)
Unit Test Result (SWE.04)
Int. Test Result (SWE.05)

Figure 1.2: Project's workflow

Conclusion

Through this chapter, we highlighted the context of our project by presenting in first place the host company shortly and its activities. We introduced the problem and the currently available solutions and we identified the objectives and contributions of this project.

Chapter 2

General overview

Introduction

This chapter summarizes the relevant fundamentals for further understanding of the proposed work. We will gain a general understanding of software development, leverage the V model, and the Automotive SPICE process model and the purpose of its application in the development of software-based systems.

2.1 What is software development?

Software is made up of a number of instructions or programs that provide computers the capacity to do specific tasks. The capacity to design and oversee computer systems is also provided. It enables flexibility and programming in computing systems by enabling computers to do a wide range of jobs and activities in line with preprogrammed instructions.

The creation and maintenance of the source code are both steps in the software development process. However, it goes much further than that, spanning every stage from conceptualizing the needed program to its actual implementation, often in line with a planned and ordered methodology that frequently overlaps with software engineering. All actions that lead to the creation of software products, including research, new development, prototyping, modification, reuse, re-engineering, maintenance, and others, are seen as being a component of software development. [5].

2.1.1 Software development process steps

Software development typically involves a series of stages and activities, which can be summarized as follows [6]:

- Methodology Selection: Choosing a suitable methodology to give a framework for the software development process, such as Agile, De- vOps, RAD, SAFe, V model, or others.
- Gathering requirements: Being aware of and recording user and stakeholder needs and expectations.
- Architecture Selection/Development: Choosing or developing the foundational framework and structure within which the program will run.
- Design Development: Producing answers to the specified criteria, sometimes with the use of process models and storyboards.
- Model Creation: Conducting early design validation, prototyping, and simulation using modeling tools and languages like SysML or UML.
- Code construction: Using the selected programming language to write the real code, with peer and team reviews to find and fix problems as soon as possible and guarantee high-quality software

- Testing: Conducting testing activities, including predefined scenarios, to verify the functionality and performance of the software.
- Configuration and Defect Management: Managing software artifacts, addressing and tracking defects, establishing quality assurance priorities, and release criteria.
- Deployment: Deploying the software for use and addressing any user problems or issues that arise.
- Data Migration: Transferring data from existing applications or sources to the new or updated software, if necessary.
- Project Management and Measurement: Monitoring and evaluating the project to ensure quality and on-time delivery, utilizing models like the Capability Maturity Model (CMM) to assess the development process.
- The software development process aligns with application lifecycle management (ALM), which includes stages like requirements analysis, design and development, testing, deployment, and maintenance and support. The lifecycle emphasizes the importance of continuous improvement, as issues identified during maintenance and support can inform requirements for future cycles.

2.1.2 Software development life cycle (SDLC)

Organizations have the freedom to select the most suitable software development life cycle (SDLC) model or alternative methodology for managing various projects, including software development. It is crucial to thoroughly study and understand the chosen approach in order to align it with the specific characteristics and risks of the project [6].

A software life cycle model can be classified as either descriptive or prescriptive, depending on its purpose. A descriptive model provides a retrospective account of the development process of a particular software system, documenting its historical progression. Descriptive models are valuable for analyzing and improving software development processes or serving as a basis for creating empirically-based prescriptive models. Conversely, prescriptive models offer recommended approaches or a set of guidelines for the development of software.

2.1.2.1 V-shaped Model

A systematic and structured approach to the software development life cycle (SDLC) is emphasized by the V-model, a software development paradigm. Its distinctive V-shaped figure, which depicts the activity flow during the development process, gave rise to its name. The V-model includes a parallel and sequential link between the testing phases and the development phases. There is a distinct testing phase connected with each stage of the software development process, creating a parallel structure, see Figure 2.1.

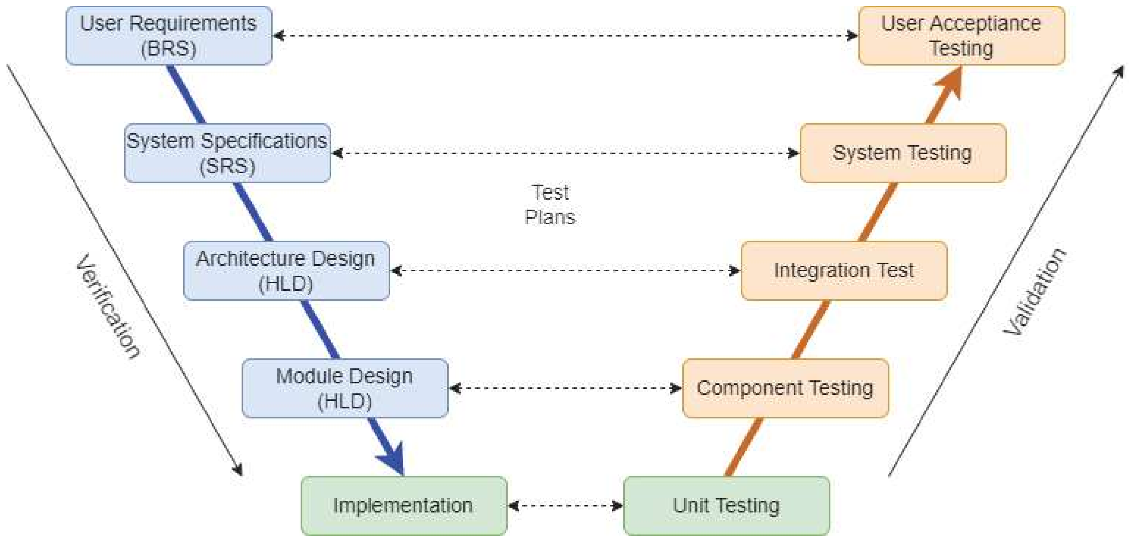


Figure 2.1: V-shaped model in software development

In this model, known as the V-model, every stage must be successfully finished before advancing to the subsequent one. The V-model prioritizes testing more heavily than other development models. Before any coding is done, testing processes are created and carried out in each phase leading up to the implementation.

Gathering needs is the first step in the life cycle. A system test plan is created prior to the start of development, and it is focused on ensuring that the functionality that was stated during requirements gathering is fulfilled.

System architecture and design are the main focus during the high-level design phase. For the purpose of assessing the interoperability of various software components, this step also includes developing an integration test strategy.

The actual software components are designed at the low-level design phase, along with unit testing. Coding activities take performed during the implementation phase. The development process proceeds along the route of execution upwards on the right side of the V-model when the coding phase is finished.

At this stage, the previously developed test plans are put into action. The testing phase encompasses executing the system and integration tests, ensuring that the software meets the defined requirements and functions effectively.

2.1.2.2 Project definition phases in a V-model

- Requirements analysis:
  Analyzing user requirements is a step in gathering product requirements. Without going into specifics of the system's architecture or implementation, this phase largely focuses on defining the required functionality of the system. A detailed explanation of the functionality, interface, performance, data, security, and other requirements as requested by the user is provided in the user requirements document. It serves as a channel of communication between users and business analysts, offering direction to system designers throughout the next design stage. At this level, user acceptability

tests are also created.

To effectively gather requirements in both software and hardware development methodologies, various methods such as interviews, questionnaires, prototypes, and use case analysis are employed. These methods facilitate the process of capturing and understanding user needs.

- System design:
  During the systems design phase, system engineers carefully examine the user requirements document to gain a comprehensive understanding of the proposed system's business aspects. They explore various possibilities and methods to effectively implement the user requirements. If any requirements are deemed impractical or unattainable, the user is promptly notified, and efforts are made to find a suitable solution. As a result, the user requirement document is updated and revised to reflect the agreed-upon resolutions.

- Architecture design:
  High-level design is the term used to describe the process of creating computer and software architecture. The main goal while choosing the architecture is to make sure it satisfies all the prerequisites. This comprises identifying the interface links and dependencies, defining the modules' core functionality, choosing the database tables, drawing architectural diagrams, and supplying technical information. During this particular stage, the design for integration testing is also developed.

- Module design:
  Low-level design is another name for the phase known as module design. In order to allow direct coding by the programmer, each module in the intended system is broken down into smaller components or modules at this step. Additionally, at this point, the unit testing design is designed.

2.1.2.3 Validation phases in a V-model

Each level of the verification phase and its equivalent stage in the validation phase are parallel in the V-model. This guarantees that the validation and verification processes are in sync. The validation step of the V-model generally consists of the following phases:

- Unit testing:
  Unit Test Plans (UTPs) are created at the module design stage of the V-Model. These UTPs are used to find and fix issues at the unit or code level. The smallest autonomous item, such a program module, is referred to as a unit. Unit testing guarantees that the tiniest object functions properly when separated from the other programs or units.

- Integration testing:
  In the architectural design phase, integration test plans are created. These test plans are designed to validate the coexistence and communication capabilities of independently created and tested units. The tests ensure that the integrated units function harmoniously together. The results of these tests are shared with the

customer's team to provide transparency and ensure alignment with the project's objectives.

- System testing:
  During the phase of System Design, the development of System Test Plans takes place, which distinguishes them from Unit and Integration Test Plans. These plans are created by the business team of the client with the objective of validating the application's alignment with the client's expectations. System Testing encompasses a comprehensive evaluation of the entire application, including its functionality, interdependencies, and communication aspects. Its primary purpose is to verify the fulfillment of both functional and non-functional requirements. To ensure the application's robustness and reliability, various subsets of System Testing, such as load and performance testing, stress testing, and regression testing, are conducted. These subsets further contribute to the assurance of a high-quality application.

- User acceptance testing:
  When conducting the requirements analysis, the development of User Acceptance Test (UAT) Plans takes place. These plans are created by business users who will be the ultimate users of the system. UAT is conducted in an environment that closely simulates the production environment, utilizing realistic data. The primary objective of UAT is to verify that the delivered system meets the user's requirements and is fully prepared for real-world usage. It serves as a conclusive confirmation that the system fulfills user expectations and is ready for deployment.

2.1.2.4 Advantages of the V-shaped models

- Straightforward and user-friendly.
- Every phase has distinct outputs or deliverables.
- Compared to the waterfall paradigm, test plans are prepared earlier in the life cycle, increasing the likelihood of success.
- Well-suited for smaller projects with clear and easily comprehensible requirements.

2.1.2.5 Disadvantages of the V-shaped models

- Highly inflexible, similar to the waterfall model.
- Limited adaptability and making changes to the scope is challenging and costly.
- Software development occurs solely during the implementation phase, with no provision for early software prototypes.
- The model lacks a well-defined approach for addressing issues discovered during testing stages.

2.2 What is Automotive SPICE?

Software Process Improvement and Capability Determination, or Automotive SPICE, is a standard of quality created to evaluate business processes from a quality and safety standpoint. It was created to evaluate, compare, and improve the processes associated with software development for the automobile industry. This standard takes into account not only software operations but also how software, hardware, and mechanics interact in a mechatronic system.

The outcome of this standard is a process model that conforms to the ISO/IEC 30xx series of standards and is acknowledged worldwide. As a result, the worldwide automobile industry is quickly adopting it as the acknowledged development standard.

2.2.1 Process reference model

There are two important components in the process reference model [7]. The first is the process aspect, which describes the defined processes and their associated requirements. Each process offers a detailed summary of the tasks that must be accomplished as part of a project. The maturity or capacity element is the second part of the model. This makes it possible to assess each process's capacity. To simplify the process dimensions, the VDA has defined 16 key processes that are fundamental to the creation of software-based systems (see Figure 2.2).

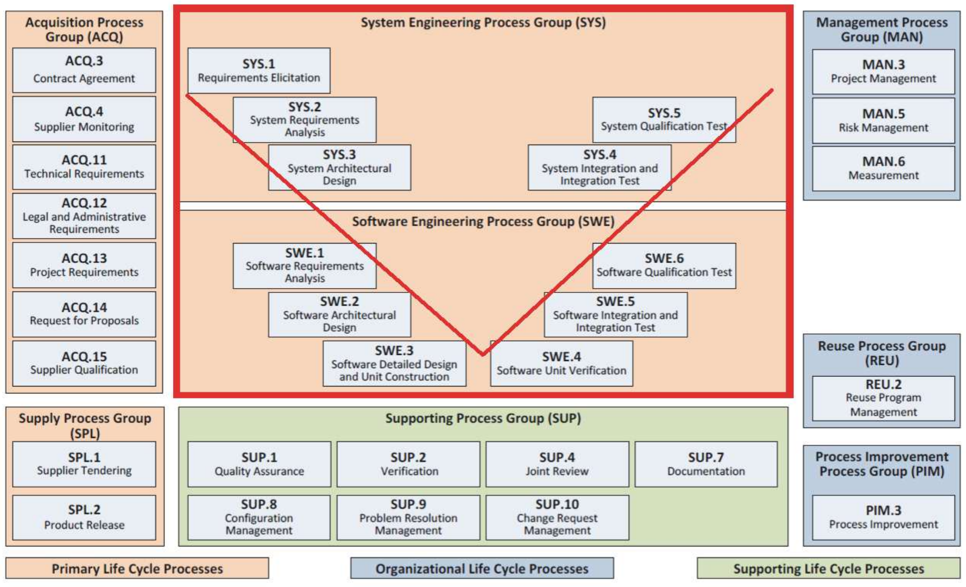

Acquisition Process Group (ACQ)
ACQ.3 Contract Agreement
ACQ.4 Supplier Monitoring
ACQ.11 Technical Requirements
ACQ.12 Legal and Administrative Requirements
ACQ.13 Project Requirements
ACQ.14 Request for Proposals
ACQ.15 Supplier Qualification
System Engineering Process Group (SYS)
SYS.1 Requirements Elicitation
SYS.2 System Requirements Analysis
SYS.3 System Architectural Design
SYS.4 System Integration and Integration Test
SYS.5 System Qualification Test
Software Engineering Process Group (SWE)
SWE.1 Software Requirements Analysis
SWE.2 Software Architectural Design
SWE.3 Software Detailed Design and Unit Construction
SWE.4 Software Unit Verification
SWE.5 Software Integration and Integration Test
SWE.6 Software Qualification Test
Management Process Group (MAN)
MAN.3 Project Management
MAN.5 Risk Management
MAN.6 Measurement
Reuse Process Group (REU)
REU.2 Reuse Program Management
Supply Process Group (SPL)
SPL.1 Supplier Tendering
SPL.2 Product Release
Supporting Process Group (SUP)
SUP.1 Quality Assurance
SUP.2 Verification
SUP.4 Joint Review
SUP.7 Documentation
SUP.8 Configuration Management
SUP.9 Problem Resolution Management
SUP.10 Change Request Management
Process Improvement Process Group (PIM)
PIM.3 Process Improvement
Primary Life Cycle Processes
Organizational Life Cycle Processes
Supporting Life Cycle Processes

Figure 2.2: Automotive SPICE process reference model

Let's now take a closer look at the process dimensions' structure. The process dimensions may first be arranged into separate categories, and then according to the activities they cover, they can be further divided into process groups.

Primary life cycle processes, organizational life cycle processes, and supporting life cycle processes are the three basic categories under which the process dimensions may be divided. Every process has a purpose statement that describes its specific goals when it is carried out in a particular setting. There is a list of precise outcomes that correspond to each purpose statement, and these outcomes indicate the intended benefits of carrying out the procedure.

2.2.1.1 Primary Life Cycle Processes

Four types of activities that are crucial for the creation of software-based systems in the automobile sector are included in the Primary Life Cycle activities Category of the automobile SPICE model. These four groups are the Software Engineering process group, the System Engineering process group, the Supply process group, and the Acquisition process group. A brief description of each of these process groups is provided below, along with an illustration of each:

- Acquisition process group (ACQ): This group of processes focuses on identifying and selecting suppliers for software-based systems. An example of a process in this group might involve developing a request for proposal to send to potential suppliers and evaluating their proposals to determine which supplier best meets the needs of the project. The ACQ processes are: Contract Agreement, Supplier Monitoring, Technical Requirements, Legal and Administrative Requirements, Project Requirements, Request for Proposals, and Supplier Qualification.
- Supply process group (SPL): This group of processes focuses on managing the relationship between the customer and the supplier of software-based systems. An example of a process in this group might involve negotiating a service level agreement (SLA) with the supplier to ensure that the software-based systems meet the specified quality and performance requirements. The SPL processes are Supplier Tendering and Product Release.
- System Engineering process group (SYS): This group of processes focuses on designing and developing the entire system that includes software, hardware, and mechanical components. An example of a process in this group might involve developing a system architecture that defines the various components of the system and how they interact with each other. The SYS processes are Requirements Elicitation, System Requirements Analysis, System Architectural Design, System Integration and Integration Test, and System Qualification Test.
- The Software Engineering process group (SWE) is responsible for the creation of software-based systems. Within this group, specific processes are carried out to develop and maintain software components. For instance, these processes can involve tasks such as writing code for a car infotainment system or conducting thorough testing to ensure the software meets the defined quality and performance criteria. The SWE processes encompass activities like Software Requirements Analysis, Software Architectural Design, Software Detailed Design and Unit Construction, Software Unit Verification, Software Integration, Integration Test, and Software Qualification Test.

2.2.1.2 Supporting Life Cycle Processes

A group of procedures that are necessary for the successful management and support of the major life cycle processes are included in the Supporting Life Cycle Procedures (SUP) category of the Automotive SPICE model. These procedures emphasize resource management, project monitoring and control, and assuring the efficient use of tools and techniques. The SUP processes include Configuration Management, Problem Resolution Management, Verification, Joint Review, Documentation, Quality Assurance, and Change Request Management.

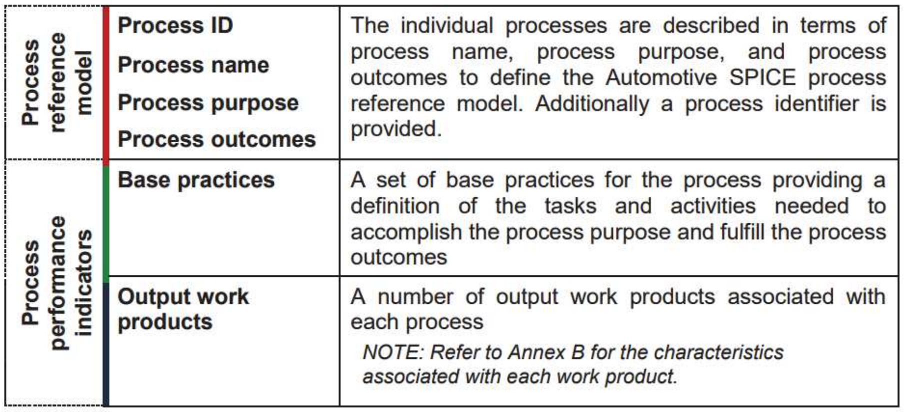
2.2.1.3 Organizational Life Cycle Processes Category

The term ”organizational life cycle processes” refers to a group of procedures that aid in the creation of process, product, and resource assets. These resources help the organization accomplish its business objectives when they are used by initiatives inside the company. Three categories make up the organizational life cycle processes category: Management process group (MAN), Process Improvement process group (PIM), and Reuse process group (REU).

2.2.2 Standard for All Processes

The description of each process follows a standardized documentation format, which includes an ID, process name, process purpose, process outcome, basic practices, and output work products (refer to Figure 2.2).

The process reference model (highlighted by a red line) and the process performance indicators used to build the process assessment model are both included in the process dimension tables, where each belongs to a particular process. The basic practices (shown by a green line) and output work products (represented by a blue line) make up the process performance indicators.



Process reference model	**Process ID** **Process name** **Process purpose** **Process outcomes**	The individual processes are described in terms of process name, process purpose, and process outcomes to define the Automotive SPICE process reference model. Additionally a process identifier is provided.
Process performance indicators	**Base practices**	A set of base practices for the process providing a definition of the tasks and activities needed to accomplish the process purpose and fulfill the process outcomes
	Output work products	A number of output work products associated with each process *NOTE: Refer to Annex B for the characteristics associated with each work product.*

Figure 2.3: Template for the process description

The following is a description of each element's significance:

- The process is distinctively recognized by its ID and process name.
- Process purpose is a brief description of the primary objective of the process that must be negotiated and agreed upon by the vendor.
- The process outcome clearly outlines the requirements, obligations, deliverables, commitments, and ultimate result of a completed process as well as the expectations of the provider and acquirer.
- Basic practices outline the specific activities required to be executed for the purpose of accomplishing the envisioned process outcome. They act as indicators for the process dimension and are the cornerstones of process evaluation.
- The output work products provide the potential deliverables that might be produced as a result of the process. It's vital to remember that this list is only recommended and not required. Depending on the circumstances, the work output may be divided among several deliverables.

2.2.3 Capability Measurement

ASPICE specifies a set of process capability levels that businesses may use to assess the effectiveness of their software development processes. These levels range from ASPICE Level 0 (Non-Compliant) to ASPICE Level 5 (Optimized). The standard also specifies a set of process qualities that must be met in order to achieve each capability level, as shown in Figure 2.3. These characteristics let enterprises analyze their adherence to Automotive SPICE and find opportunities for improvement.

- Level 0: Basic. The process is either not conducted or does not achieve its intended goal. At this point, organizations can only achieve ASPICE criteria to a certain extent and should concentrate on managing vital activities over attaining higher standards.
- Level 1: Performed. The process is implemented and achieves its intended goal. Organizations at this point can deliver standard requirements to a significant extent or completely, although there may still be some gaps in their processes.
- Level 2: Managed. The process is managed, as well as the work products are developed, monitored, and maintained. Organizations at this point consistently deliver work products and achieve ASPICE standards to a significant extent or completely.
- Level 3: Established. A defined process is used based on a standard process. At this point, organizations have set performance standards for the entire organization and continually evaluate and learn from them.

- Level 4: Predictable. The process is consistently enacted within defined limits. A defined process is used based on a standard process. In addition to establishing and meeting performance standards, organizations at this level measure, record, and analyze outcomes to enable objective evaluation.

- Level 5: Optimizing. The process is continuously improved to meet current and projected business goals. At this highest level, organizations not only achieve and analyze performance standards for quantitative feedback and causal analysis resolutions but also invest in continuous improvement.

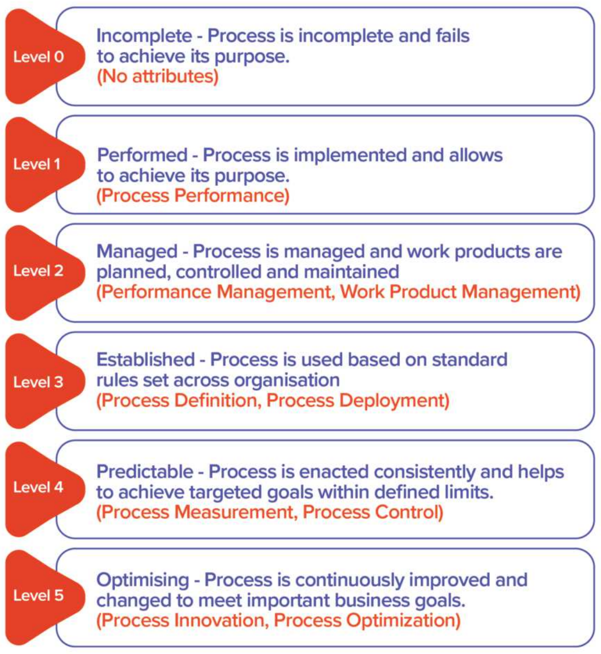

Level 0
Incomplete - Process is incomplete and fails to achieve its purpose.
(No attributes)
Level 1
Performed - Process is implemented and allows to achieve its purpose.
(Process Performance)
Level 2
Managed - Process is managed and work products are planned, controlled and maintained
(Performance Management, Work Product Management)
Level 3
Established - Process is used based on standard rules set across organisation
(Process Definition, Process Deployment)
Level 4
Predictable - Process is enacted consistently and helps to achieve targeted goals within defined limits.
(Process Measurement, Process Control)
Level 5
Optimising - Process is continuously improved and changed to meet important business goals.
(Process Innovation, Process Optimization)

Figure 2.4: Maturity levels in ASPICE[8]

2.2.4 Benefits of ASPICE

Implementing ASPICE for software development in the automotive industry offers several advantages. Some of them are:

- Enhanced Software Quality: ASPICE provides a structured framework for evaluating and enhancing software development processes. This enables the identification and elimination of inefficiencies, reducing the occurrence of defects and errors in software development see figure 2.4.

- Improved Efficiency: By adopting ASPICE, organizations can streamline their software development processes, resulting in reduced time and resources required for software development and maintenance. This leads to cost savings and increased productivity.

- Enhanced Communication: ASPICE establishes a common language and a set of expectations for software development processes across the industry. This fosters better communication among suppliers, manufacturers, and other stakeholders involved in the software development lifecycle.
- Heightened Customer Satisfaction: The improved quality and reliability of software developed under ASPICE can significantly enhance customer satisfaction. This, in turn, contributes to a positive brand reputation, customer loyalty, and increased business success.
- Compliance with Industry Standards: Adhering to ASPICE helps organizations comply with industry standards and regulations, demonstrating their commitment to delivering high-quality software. This ensures the safety and reliability of their products, aligning with the expectations of customers and regulatory bodies.

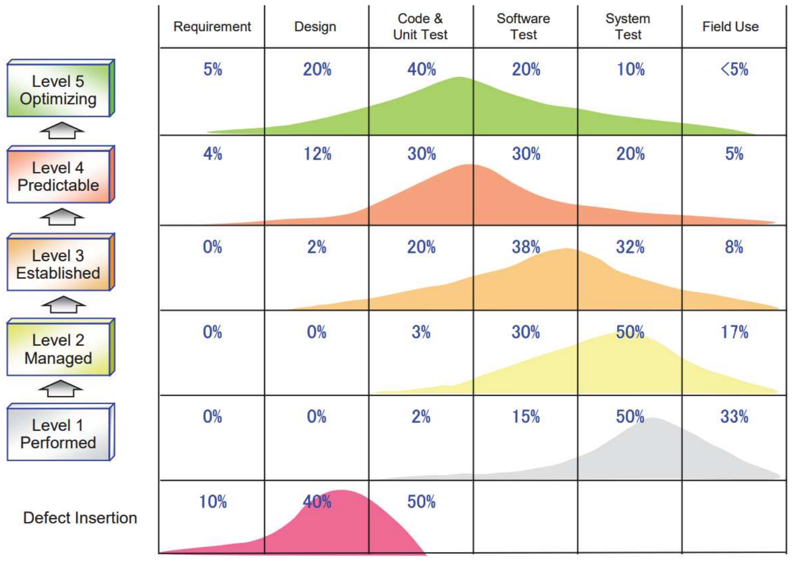


Figure 2.5: Early defect detection

2.2.5 Challenges with ASPICE

While ASPICE offers valuable advantages for software development in the automotive industry, it does present certain challenges. Here are some common challenges associated with ASPICE implementation:

- Complexity: ASPICE is a comprehensive and intricate framework that can be challenging for organizations to comprehend and implement. Its complexity may lead to resistance from team members and require additional time and resources for successful adoption.
- Resource Constraints: Implementing ASPICE demands substantial resources, including investments in training, tools, and processes. This can pose challenges, especially for smaller organizations with limited resources.

- Resistance to Change: Implementing ASPICE necessitates significant changes to an organization's existing processes and practices. Resistance to change from team members or stakeholders can impede the smooth implementation of the framework.
- Lack of Industry Standardization: Although ASPICE is widely used in the automotive industry, there is still a lack of standardization across different companies and organizations. Inconsistencies in its application can arise, making it challenging for suppliers to meet the requirements of multiple customers.
- Integration with Existing Processes: ASPICE must be integrated with an organization's existing processes, tools, and methodologies. This can pose difficulties, particularly if the organization has already invested in tools and processes that may not align seamlessly with the ASPICE framework.

2.2.6 ASPICE Vs ISO-26262

ASPICE and ISO 26262 are two standards relevant to the automotive industry, although they differ in their scope and focus [8]. ASPICE, as previously mentioned, is a process assessment model specifically designed to evaluate and enhance software development processes within the automotive sector. It encompasses the entire software development lifecycle and emphasizes an organization's process capabilities.

In contrast, ISO 26262 is a safety standard that provides guidelines to ensure the functional safety of electrical and electronic systems in vehicles. It outlines requirements for safety management, hazard analysis and risk assessment, safety verification and validation, and other aspects. The primary emphasis of this standard is on effectively managing and controlling safety risks associated with the utilization of such systems in vehicles.

While ASPICE focuses on the software development process and its capabilities, ISO 26262 is primarily concerned with the safety aspects of electrical and electronic systems employed in vehicles. These two standards complement each other, and many organizations involved in automotive software development must comply with both.

To summarize, ASPICE offers a framework for assessing and improving software development processes, while ISO 26262 provides guidelines for ensuring the safety of electrical and electronic systems in vehicles.

Conclusion

Through this chapter, we have clarified the different concepts related to ASPICE to improve software quality and reliability. That will help us to better undertake the project and understand the workflow. The next chapter will cover the details of the data analysis and retrieval to implement our solution.

Chapter 3

KPI Analysis and Data Retrieval

Introduction

The following chapter will present an in-depth analysis of the key KPIs chosen for the dashboard and elucidate the techniques employed to retrieve and integrate the requisite data. By examining the KPIs and data retrieval process, this chapter aims to establish a solid foundation for understanding how the dashboard will provide meaningful insights and facilitates data-informed decision-making processes.

3.1 Key Performance Indicators (KPIs) selection

3.1.1 What is a KPI?

A Key Performance Indicator (KPI) is a quantifiable measure that signifies the performance and advancement of an organization or a particular aspect of its operations. KPIs serve the purpose of assessing the attainment of strategic objectives, monitoring progress toward specific goals, and offering valuable insights into the efficiency of different processes, activities, or initiatives.

The selection of KPIs is done with great care to align with the critical success factors and goals of the organization. They play a vital role in monitoring performance, identifying areas that require improvement, and facilitating data-driven decision-making. KPIs can take the form of quantitative or qualitative measures and are typically established using specific metrics or benchmarks.

3.1.1.1 Types of KPIs

Most Key Performance Indicators can be categorized into distinct groups, each with its own unique characteristics, time frame, and intended users [9]:

- Financial KPIs: KPIs that revolve around finance primarily focus on revenue and profit margins, particularly net profit. Examples of financial KPIs include liquidity ratios, profitability ratios, solvency ratios, and turnover ratios.

- Customer Experience KPIs: These KPIs primarily concentrate on the efficiency, satisfaction, and retention of customers before they become actual customers. Customer service teams utilize these metrics to gain a better understanding of the service customers receive. Examples of customer experience KPIs include the number of new ticket requests, number of resolved tickets, average resolution time, average response time, top customer service agent, type of request, and customer satisfaction rating.

- Process Performance KPIs: Process metrics are used to measure and monitor operational performance within an organization. These KPIs assess task performance and identify process, quality, or performance issues. They are particularly valuable for companies with repetitive processes or those in cyclical industries. Examples of process performance KPIs include production efficiency, total cycle time, throughput, and error and quality rate.

- Marketing KPIs: Marketing KPIs are utilized to assess the effectiveness of marketing and promotional campaigns, focusing on measuring conversion rates and the actions taken by potential customers in response to marketing efforts. Examples of marketing KPIs include website traffic, social media traffic, the conversion rate on call-to-action content, and click-through rates.

- IT KPIs: In pursuit of operational excellence, companies may monitor the performance of their internal technology (IT) department. IT KPIs and metrics related to hardware, software, and other internal technology aspects are used. Examples of IT KPIs include total system downtime, number of tickets/resolutions, number of developed features, count of critical bugs, and backup frequency.

- Sales KPIs: Generating revenue through sales is a primary objective for companies. While financial KPIs measure revenue, sales KPIs offer a more detailed analysis using non-financial data to gain insights into the sales process. Examples of sales KPIs include customer lifetime value (CLV), customer acquisition cost (CAC), the average dollar value for new contracts, average conversion time, and the number of engaged leads.

- Human Resource and Staffing KPIs: Analyzing employee-specific KPIs can be beneficial for companies, allowing them to delve into metrics related to turnover, retention, and satisfaction, given the abundance of available information about their staff. Examples of human resource or staffing KPIs include absenteeism rate, number of overtime hours worked, employee satisfaction, employee turnover rate, and number of applicants.

3.1.1.2 Advantages of KPIs

The benefits of using Key Performance Indicators in a business are numerous, and a company would like to analyze KPIs for several reasons. In fact, KPIs serve as valuable tools for management, offering quantitative insights into specific issues and aiding in strategic planning to achieve a certain objective. Furthermore, KPIs act as a vital link between business operations and overarching goals. While companies may establish targets, it is the ability to track progress through KPIs that gives purpose to those plans. KPIs enable companies to set objectives and systematically monitor their advancement, ensuring alignment with different goals.

3.1.1.3 Challenges with KPIs

Many challenges can arise in the process of defining and measuring KPIs, of which we can mention long-time framed KPIs. These KPIs require a long time frame to be collected and used by the company to provide meaningful insights on trends and satisfaction rates over a long period of time. Without ongoing monitoring and diligent follow-up, a KPI is unuseful. Simply preparing a KPI report without subsequent analysis is meaningless and doesn't help in decision-making.

3.1.2 Selection of KPIs for ASPICE SWE assessment

KPIs play a vital role in the dashboard by providing quantifiable measurements to assess the effectiveness of ASPICE implementation. The main focus of this project is to provide a dashboard incorporating ASPICE SWE KPIs to assess the state of ASPICE compliance during the process of software development.

The dashboard exclusively focuses on Software Engineering (SWE) metrics, as it is specifically tailored to track and monitor the performance of software development activities within our team see Figure 3.1.

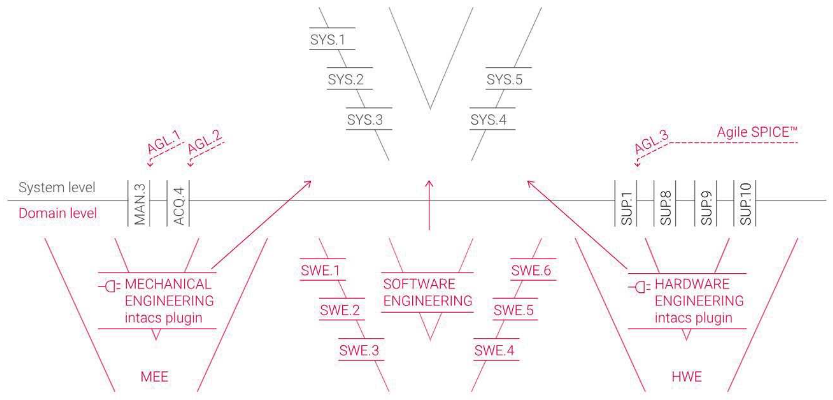


Figure 3.1: Automotive SPICE as part of VDA Scope [10]

Software Engineering (SWE) consists of six essential components that play a crucial role in ensuring the success and quality of software development processes. Within each component of SWE, we have carefully curated a collection of KPIs and metrics. These indicators and metrics offer valuable insights and provide a comprehensive view of each step in the software development process. By monitoring and analyzing these KPIs, we gain a deeper understanding of the progress, performance, and quality of our software development efforts.

3.1.2.1 SOFTWARE REQUIREMENTS ANALYSIS - SWE.1 KPIs

The process of software requirements analysis in Automotive SPICE involves converting the software-related aspects of system requirements into a set of software requirements, as stated in [11].

The documentation of software requirements is essential for several reasons. While system or customer requirements may already be in place, investing time and effort into documenting additional software requirements serves important purposes. The ultimate goal of any project is to deliver the agreed-upon results within the allocated timeframe, and budget, and meet the customer's quality standards. By documenting software requirements, you reduce the risk of overlooking crucial functionality or misinterpreting the customer's expectations. Neglecting to document these requirements can lead to unexpected costs, delays, and potential misunderstandings. Furthermore, without proper documentation, you may overlook critical aspects of your software, both in terms of functionality and non-functional requirements. This oversight can result in false starts or the need for additional development cycles, leading to further delays and resource consumption. Therefore, the documentation of software requirements is crucial to ensure accurate understanding, effective communication, and successful outcomes in software development.

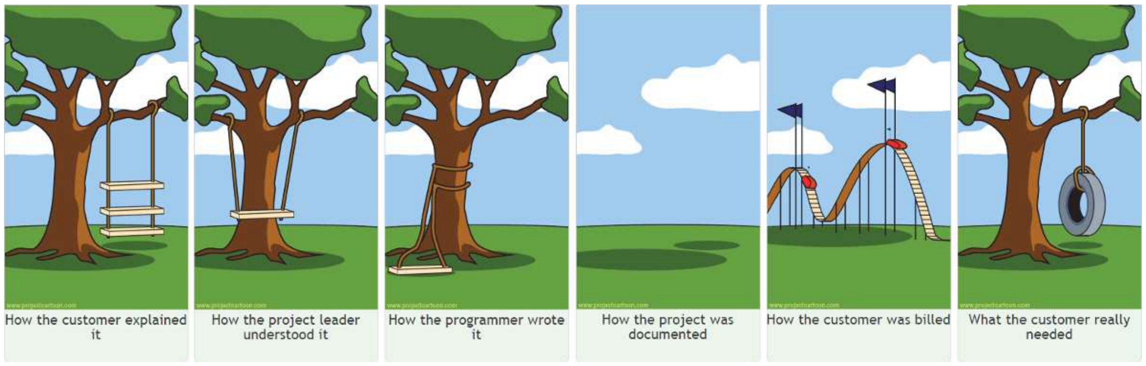

How the customer explained it
How the project leader understood it
How the programmer wrote it
How the project was documented
How the customer was billed
What the customer really needed

Figure 3.2: Importance of Software requirements documentation

The most important aspect of software requirements is that you need to consider more than your customer's expectations. Every software has to meet standards, norms, and other regulations that are considered as requirements.

The software requirements describe the software as a black box, it answers the "What?" and not the "How?" and according to Automotive SPICE, adherence to certain base practices is essential to ensure compliance with industry standards and achieve desired quality outcomes. Some of these base practices we can mention:

- **BP1:** Specify software requirements by utilizing system requirements, system architecture, and any changes to them. This process involves identifying the necessary functions and capabilities of the software, both functional and non-functional, and documenting them in a software requirements specification.
- **BP2:** Structure software requirements by organizing them in the software requirements specification using various techniques, such as: Grouping them into clusters that are relevant to the project.
- **BP3:** Analyze software requirements to ensure their correctness, technical feasibility, and verifiability, and to identify any risks associated with them. Analyze the impact of the requirements on cost, schedule, and technical aspects.
- **BP4:** Evaluate the influence of the software requirements on system interfaces and the operational environment. Examine the implications of the requirements for the interactions between software, hardware components, operating systems, and other elements within the system.

- **BP5:** Define verification criteria. Define the criteria for verifying each software requirement, including qualitative and quantitative measures. These criteria serve as the basis for developing software test cases or other verification measures to ensure compliance with the requirements.

- **BP6:** Ensure bidirectional traceability. Establish a bidirectional relationship between system requirements and software requirements. Establish a bidirectional connection between the system architecture and software requirements. Bidirectional traceability enables coverage, consistency, and impact analysis.

- **BP7:** Maintain consistency. It is essential to maintain consistency between the system requirements and software requirements, as well as between the system architecture and software requirements. Consistency can be achieved through bidirectional traceability and can be demonstrated through review records.

- **BP8:** Communicate agreed software requirements. Share and communicate the agreed software requirements, as well as any updates or changes to the software requirements, with all relevant stakeholders.

Now that we have discussed the base practices related to software requirements analysis (SWE.1), let's delve into the key performance indicators associated with this component:

Software requirements status KPI:

To effectively monitor the overall project status, it is recommended to continuously track the progress of each requirement, as mentioned in [12]. This can be achieved by considering the use of a requirement attribute specifically designed to store this information. By actively monitoring requirement statuses, you can address the common challenge of software projects often being perceived as ”ninety percent done” indefinitely. At any given time, each requirement can be categorized into one of the following statuses:

- Proposed: The requirement was suggested by someone.
- Approved: The requirement has been allocated to a baseline.
- Implemented: The code for the requirement has been designed, written, and unit tested.
- Verified: The requirement has successfully passed its tests after being integrated.
- Deferred: The requirement is scheduled for implementation in a future release.
- Deleted: The decision has been made not to implement the requirement at all.
- Rejected: The idea behind the requirement was never approved.

SWE.01	No. Of SWRQ									
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
in work	10	25	35	30	25	15	10	7	0	0
proposed	12	19	25	30	30	35	35	21	20	0
approved	0	5	5	5	10	15	20	35	43	63
rejected	0	5	5	5	5	5	5	7	7	7

Table 3.1: Software requirements status KPI

When inquiring about the progress of a project, receiving detailed information from your employees can be more enlightening than vague statements like ”I’m about ninety percent done. Lookin’ good!” They might instead provide specific details such as, ”Out of the eighty-seven requirements assigned to this subsystem, sixty-one have been verified, nine have been implemented but not yet verified, and seventeen are still in the process of being fully implemented.”

To visually represent the completeness of a project, a graph depicting the status of requirements can be immensely helpful. This graph showcases the distribution of requirement statuses, offering valuable insights into the progress made and the remaining work. By examining this visual representation, stakeholders can better understand the overall project status and evaluate the level of effort needed to successfully complete the project.

The graph effectively highlights various requirement statuses, including ”Proposed,” ”Approved,” ”Implemented,” and ”Deferred.” Each status provides specific information about the state of the corresponding requirement. For instance, if a significant portion of requirements is marked as ”Not Started,” it indicates that there is substantial work ahead, necessitating the allocation of more effort to initiate those requirements. Conversely, a higher proportion of requirements labeled as ”Completed” signifies progress and suggests that the project is advancing toward its intended goals.

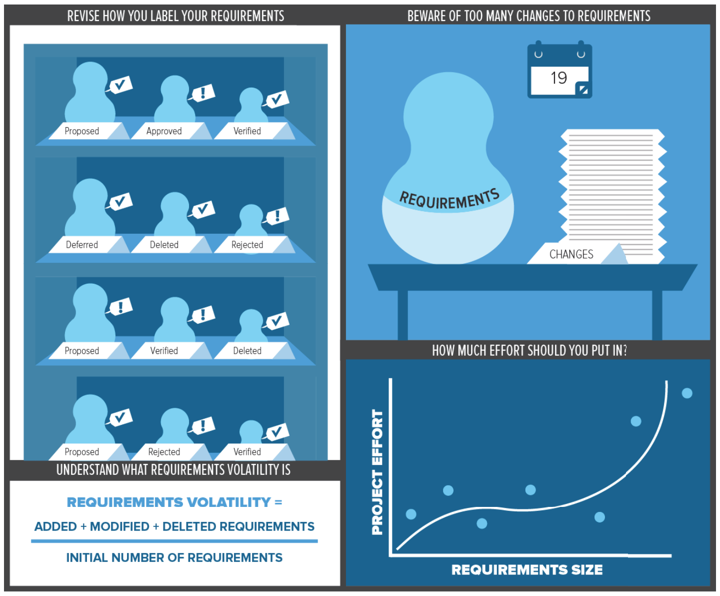

REVISE HOW YOU LABEL YOUR REQUIREMENTS
Proposed
Approved
Verified
Deferred
Deleted
Rejected
Proposed
Verified
Deleted
Proposed
Rejected
Verified
UNDERSTAND WHAT REQUIREMENTS VOLATILITY IS
REQUIREMENTS VOLATILITY =
ADDED + MODIFIED + DELETED REQUIREMENTS
INITIAL NUMBER OF REQUIREMENTS
BEWARE OF TOO MANY CHANGES TO REQUIREMENTS
19
REQUIREMENTS
CHANGES
HOW MUCH EFFORT SHOULD YOU PUT IN?
PROJECT EFFORT
REQUIREMENTS SIZE

Figure 3.3: Measuring requirements status

Traced Requirement:

By implementing Traceability, we define relations of different types between development artifacts such as requirements, architecture, implementation design, tests, issues, and change requests.

SWE.01	Traced Requirement
Total SWRQ	350
With upward trace	230
Without upward trace	120

Table 3.2: Traced requirements KPI

Ensuring that each business need is linked to a specific requirement and that each requirement is connected to a tangible deliverable is the essence of requirements traceability. This practice holds great significance for business analysts as it establishes connections between requirements, solution components, and other related artifacts [13].

By implementing traceability, you gain the capability to navigate through these artifacts and conduct coverage or impact analyses. This provides a comprehensive view of your project's status, as depicted in Figure 3.4. Initially driven by the necessity to comply with safety standards like IEC-61508 and ISO26262, traceability is now increasingly acknowledged and adopted across various industries as a valuable tool for enhancing product quality and reliability.

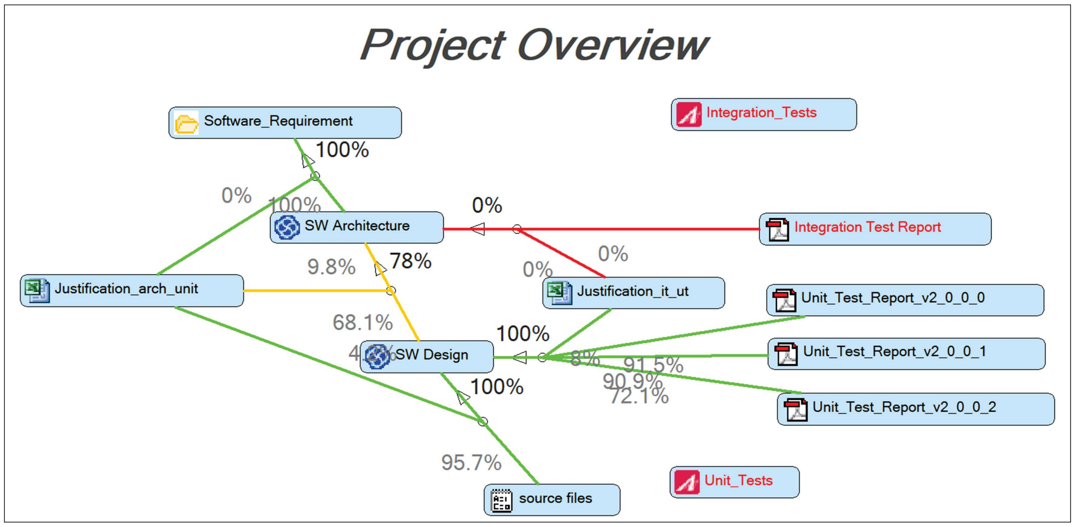


Figure 3.4: Software development process traceability

Traceability involves the systematic tracking of requirements throughout the entire product development life-cycle. It serves as a documented thread that offers both forward and backward visibility into all the activities associated with each requirement (including design, development, testing, and support) check figure 3.4. Bidirectional traceability refers to the capability of tracing information in both forward and backward directions. This means being able to trace from a requirement to a corresponding test case as well as from a test case back to the requirement. Establishing bidirectional traceability establishes a relationship between two artifacts, allowing seamless navigation between them. This ability to trace from one item to another and back again is crucial for maintaining a comprehensive understanding of the relationships, dependencies, and coverage within a project.

VD Assigned Requirement:

The term ”verification domain assigned requirements” describes the action of tying particular requirements to a specific verification domain. In other words, it entails classifying or organizing the requirements according to the component or area of the system to which they apply.

It is simpler to coordinate and organize the verification efforts when requirements are assigned to various verification domains. There may be a unique collection of tests, processes, and standards for each verification domain's associated requirements verification. With the help of this strategy, a targeted and orderly verification process can be carried out, guaranteeing that all requirements for each domain are correctly verified and validated.

Assigning requirements to verification domains facilitates effective tracking and administration of verification efforts as well as complete system coverage. Additionally, it promotes traceability.

SWE.01	VD Assigned Requirement
Total SWRQ	350
With VD Assigned	254
Without VD Assigned	96

Table 3.3: VD assigned requirements KPI

Not Analysed CR/PR:

A change request (CR) is an official proposal put forth by a stakeholder involved in a project to suggest modifications to the project's scope, deliverables, schedule, or resources. It is a documented formal request that outlines the desired changes, the reasoning behind them, and the potential impacts they may have on the project. Change requests are typically utilized when there is a need to alter the original project plan due to various factors such as new requirements, unforeseen circumstances, or evolving stakeholder needs. These requests undergo a thorough evaluation, analysis, and approval process conducted by the change control board (CCB) to assess their feasibility, impact, and alignment with the project's objectives. Once approved, a change request leads to adjustments in project activities, timelines, budgets, or other relevant aspects [13]. Monitoring the number of CRs assists in prioritizing and managing customer requirements, ultimately resulting in a higher level of customer satisfaction.

A problem report (PR) is a formal document that identifies and describes an issue or problem encountered in a product, system, or project. It is commonly employed in software development and quality control processes. A problem report is created whenever a problem or flaw is discovered during testing or actual production use. The report provides details about the nature of the problem, its impact on the product's usability or performance, steps to replicate the issue, and any other pertinent information.

Problem reporting plays a crucial role in the problem-solving and troubleshooting process. It aids in identifying and documenting issues, assigning tasks to the appropriate individuals or teams for resolution, and keeping track of the steps taken to address the concerns [14]. Tracking the number of problem reports helps identify the overall quality of the software by highlighting areas that require improvement or bug fixing. A high number of problem reports may indicate potential software defects or functionality gaps that need to be addressed.

By monitoring and tracking problem reports and change requests, project managers gain valuable metrics to assess the progress and health of the software development project. These metrics aid in resource allocation, task prioritization, and identification of potential bottlenecks or risks. Tracking these metrics enables project managers to make informed decisions and take appropriate actions to keep the project on track.

SWE.01	Not Analysed CR/PR								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
Total CR/PR	7	12	12	15	17	20	24	30	35	38
With CCB decision	0	4	4	4	6	8	12	18	20	20
Without CCB Decision	7	8	8	11	11	12	12	12	15	18

Table 3.4: VD assigned requirements KPI

The workflow of both CRs and PRs is shown in figure 3.5

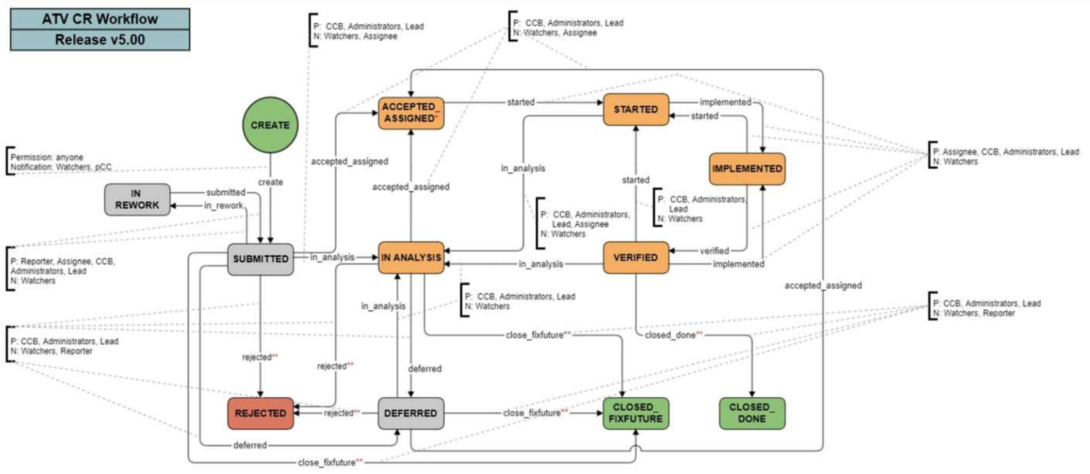


Figure 3.5: CRs and PRs workflow

3.1.2.2 SOFTWARE ARCHITECTURAL DESIGN - SWE.2 KPIs

The objective of Software architecture is to establish how the intended functionality, as outlined in the Software requirements, will be realized. While the requirements outline the desired outcomes, the architecture focuses on the approach and methodology for implementing the functionality. In essence, the requirements specify the ”what” while the architecture defines the ”how” [15].

The aim of the Software Architectural Design Process is to create a comprehensive architectural design, allocate specific software requirements to corresponding software elements, and assess the software architectural design based on predefined criteria. According to ASPICE, the base practices of software architectural design are:

- BP1: Develop and record the software architectural design, which outlines the software elements in accordance with both functional and non-functional software requirements.
- BP2: Allocate software requirements: Assign the software requirements to the components within the software architectural design.
- BP3: Define interfaces of software elements. Identify, develop and document the interfaces of each software element.
- BP4: Describe dynamic behavior: Assess and document the timing and interactive behavior of software components to ensure they meet the specified dynamic behavior of the system.
- BP5: Establish resource consumption objectives: Identify and document the resource consumption objectives for relevant elements within the software architectural design at the appropriate hierarchical level. These objectives typically include considerations such as memory usage (ROM, RAM, external/internal EEPROM, or Data Flash) and CPU load.
- BP6: Assess alternative software architectures: Establish evaluation criteria for the architecture and evaluate different software architectures based on the defined criteria. Document the reasoning behind selecting the chosen software architecture. Evaluation criteria may encompass quality characteristics (modularity, maintainability, expandability, scalability, reliability, security realization, and usability) as well as the outcomes of make-buy-reuse analysis.
- BP7: Establish bidirectional traceability: Establish bidirectional traceability between software requirements and elements within the software architectural design. This traceability ensures that there is a clear connection and understanding of how software requirements are fulfilled by the architectural elements.
- BP8: Maintain consistency: Ensure that there is consistency between the software requirements and the software architectural design. This alignment helps to ensure that the design accurately reflects and satisfies the specified requirements. Consistency can be supported by establishing bidirectional traceability and can be demonstrated through review records.
- BP9: Communicate the software architectural design: Effectively communicate the agreed-upon software architectural design and any updates to all relevant stakeholders. This ensures that all parties involved are aware of the design decisions and can provide their input or feedback as needed. Clear communication helps to foster understanding and collaboration throughout the development process.

Now that we have discussed the base practices related to software architectural design (SWE.2), let's delve into the key performance indicators associated with this component:

SWA Coverage :

Software architecture coverage provides confidence that the software architecture is comprehensive, aligned with the requirements, and capable of delivering the intended functionality and quality. By evaluating software architecture coverage, development teams can identify any gaps or inconsistencies between the requirements and the architectural design early in the development process. This allows for timely adjustments, refinements, and improvements to ensure that the architecture fully addresses the needs and expectations of the software system.

ASPICE requires bi-directionality to test the SWA coverage to ensure that architectural design adequately supports the desired system requirements and vice-versa, check Table 3.5.

SWE.02	SWA Coverage
No. Of SWRQ (not rejected)	100
Traced SWRQ (from SWA)	55
SWA Total	300
SWA Traced to SWRQ	155

Table 3.5: SWA Coverage KPI

No. Of SWA:

Tracking the status of software architectures provides visibility into the progress and status of ongoing architectural design activities. It enables team members, project managers, and stakeholders to see precisely which architectures are being developed, which ones are being considered for approval, and which ones have already been rejected. This encourages accountability and transparency throughout the development process.

SWE.02	No. Of SWA								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
in work	10	25	35	30	25	15	10	7	0	0
proposed	12	19	25	30	30	35	35	21	20	0
approved	0	5	5	5	10	15	20	35	43	63
rejected	0	5	5	5	5	5	5	7	7	7

Table 3.6: No. Of SWA KPI

3.1.2.3 SOFTWARE DETAILED DESIGN AND UNIT CONSTRUCTION – SWE.3

The Automotive SPICE Software Detailed Design and Unit Construction process also referred to as SWE.3, assists the organization in producing a thoroughly evaluated detailed design for software components and in defining and constructing software units [16]. The precise purpose of software design is often challenging for organizations to grasp. It involves capturing the intricate design of the software system. In this process, there are a few essential best practices (BP) that need to be followed:

- BP1: Develop detailed software design. Create a comprehensive design for each software component outlined in the software architecture, specifying all software units based on functional and non-functional software requirements.
- BP2: Define software unit interfaces. Identify, define, and document the interfaces for each software unit.
- BP3: Describe dynamic behavior. Assess and document the dynamic behavior and interactions among relevant software units. Note that not all software units may have dynamic behavior requiring description.
- BP4: Evaluate software design. Evaluate the software design with respect to interoperability, interaction, criticality, technical complexity, risks, and testability. The evaluation results can be utilized for software unit verification.
- BP5: Establish bidirectional traceability. Establish bidirectional traceability between software requirements and software units. Also, establish bidirectional traceability between the software architectural design and the detailed software design.
- BP6: Maintain consistency. Ensure that there is consistency between the software requirements and the software units. Additionally, ensure consistency between the software architectural design, the detailed software design, and the software units.
- BP7: Communicate approved software detailed design. Effectively communicate the approved software's detailed design and any updates to all relevant parties.
- BP8: Implement software units. Create and document the executable representations of each software unit in accordance with the software's detailed design.

SWUD Coverage :

By tracking SWUD coverage, we can ensure that each software unit has been designed and implemented to address specific software requirements. It helps verify that all requirements have been adequately covered by the software units, ensuring completeness and reducing the risk of missing any critical functionality.

SWE.02	No. Of SWA								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
in work	10	25	35	30	25	15	10	7	0	0
proposed	12	19	25	30	30	35	35	21	20	0
approved	0	5	5	5	10	15	20	35	43	63
rejected	0	5	5	5	5	5	5	7	7	7

Table 3.7: SWUD Coverage KPI

Tracking software unit design bidirectional coverage ensures alignment between requirements and implementation, promotes traceability and accountability over time, facilitates change management, and supports long-term maintenance and evolution of the software system.

Number Of SWUD :

By tracking the status of software unit design every week, we can monitor the progress of each unit's development. It provides visibility into which units are currently being worked on, which are awaiting approval, and which have been rejected. This information helps project managers and stakeholders stay informed about the overall progress of the software development process.

SWE.03	No. Of SWUD								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
in work	10	25	35	30	25	15	10	7	0	0
proposed	12	19	25	30	30	35	35	21	20	0
approved	0	5	5	5	10	15	20	35	43	63
rejected	0	5	5	5	5	5	5	7	7	7

Table 3.8: Number Of SWUD KPI

SWDI Coverage :

Tracking the software design implementation coverage provides traceability between the design and the software units. It allows you to establish a clear relationship between the design specifications and the implemented code, enabling impact analysis. This traceability helps in understanding the implications of design changes or modifications on the implemented units, facilitating change management and ensuring consistency throughout the development process.

SWE.03	SWDI Coverage
SWUD Total	500
SWUD Traced (from SWDI)	325

Table 3.9: SWDI Coverage KPI

Cyclometric Complexity :

The cyclomatic complexity (CC) metric is utilized to quantify the number of distinct paths that can be taken through a section of code [17]. Code segments with lower cyclomatic complexity tend to be more comprehensible and carry less risk when it comes to making modifications. To calculate the cyclomatic complexity, a Control Flow Graph is constructed for the code, which captures the number of independent paths that can be traversed within a program module (refer to Figure 3.6 for an illustration).

The complexity M is then defined as $M = E - N + 2 * P$ where:
E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

SWE.03	Cyclomatic complexity			
	[<=5]	[6 - 10]	[11 - 15]	[>15]
Count	350	60	2	0

Table 3.10: Cyclometric Complexity KPI

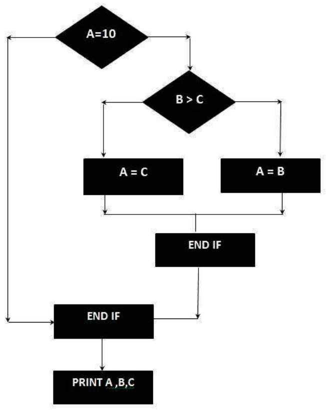


Figure 3.6: Control Flow Graph [18]

The above control flow diagram depicted in Figure 3.6 is utilized to calculate the cyclomatic complexity, which consists of seven nodes and eight edges. As a result, the cyclomatic complexity is determined by the formula 8 - 7 + 2, resulting in a value of 3.

Cyclomatic complexity serves various purposes in software development, making it a valuable tool. It functions as a quality metric, allowing for comparisons of different designs based on their relative complexity. By measuring the minimum effort required and identifying optimal areas for testing, it facilitates a streamlined testing process with comprehensive coverage. Another advantage is its ease of application, making it a convenient choice for software development teams.

MISRA Compliance :

MISRA is a set of C and C++ coding standards and guidelines, developed by the Motor Industry Software Reliability Association (MISRA). MISRA ensures that C/C++ code is safe, secure, and reliable.

C and C++ are highly popular languages utilized in embedded software development. C, specifically, has been implemented for almost every processor, offering extensive resources and libraries. It is supported by a wide range of tools and benefits from a large pool of skilled developers. However, even a C program that fully complies with the ISO language standard can still contain code with unpredictable behavior, which is unacceptable in critical applications such as a car braking system [19]. To mitigate these risks, the MISRA coding guidelines aim to impose restrictions on the usage of the language, significantly reducing potential dangers [20].

Misra compliance information is extracted from Polyspace Bug Finder which identifies software bugs, concurrency issues, run-time errors, and other C and C++ source code defects.

SWE.03	MISRA Compliance
Total Warnings	577
Unjustified Warnings	68
Justified Warnings	509

Table 3.11: MISRA Compliance KPI

Polyspace Compliance :

Polyspace, a software product created by MathWorks, is a static code analysis tool employed to identify vulnerabilities and critical run-time errors in C, C++, and other programming languages. It also ensures that your source code adheres to the required code standards by conducting comprehensive proof-checking.

Ployspace compliance information is extracted from Polyspace Code Prover which looks for run-time issues including overflow, buffer overrun, division-by-zero, out-of-bounds array access, and others in the correctness of C and C++ source code. Every code instruction is verified by the tool, which also offers a formal diagnostic for each operation in both typical and unusual usage scenarios.

SWE.03	Polyspace Compliance		
	Red Violation	Orange Violation	Grey Violation
Unjustified	2	8	7
Justified	3	12	8

Table 3.12: Polyspace Compliance KPI

Binary size :

The binary size of a microcontroller refers to the size of the compiled firmware or software that is programmed into the microcontroller's memory. It represents the amount of storage space required to store the compiled code and any associated data or resources on the microcontroller like ROM, IRAM, DRAM, and STACK. The binary size of a microcontroller program is typically measured in bytes and can vary depending on the complexity of the code, the features and libraries used, and the available memory capacity of the microcontroller. Managing the binary size is crucial in microcontroller development to ensure efficient memory utilization and optimize the performance of the embedded system.

SWE.03	Binary size								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
ROM	23%	23%	25%	25%	30%	30%	40%	40%	40%	40%
IRAM	56%	65%	75%	75%	80%	80%	90%	93%	94%	97%
DRAM	25%	32%	75%	75%	75%	75%	75%	78%	80%	82%
STACK	10%	12%	68%	68%	68%	68%	70%	72%	80%	82%

Table 3.13: Binary size KPI

3.1.2.4 SOFTWARE UNIT VERIFICATION – SWE.4 in Automotive SPICE KPIs

The Software Unit Verification process in Automotive SPICE, also referred to as SWE.4, enables organizations to ensure that software units meet the specified level of quality by implementing detailed design and non-functional requirements [21]. Unit verification plays a crucial role in the testing process and should not be overlooked. Neglecting unit verification can lead to various undesirable consequences. To prevent such issues, the following fundamental practices should be followed:

- BP1: Formulate a comprehensive software unit verification strategy, encompassing a regression strategy for re-verification in case of software unit modifications. The verification strategy should provide evidence of compliance between the software units, software detailed design, and non-functional requirements.
- BP2: Establish criteria for unit verification that effectively demonstrate compliance of the software units and their interactions within the component with the software's detailed design and non-functional requirements. When conducting unit testing, the criteria should be defined in a unit test specification.
- BP3: Conduct static verification of software units to ensure correctness, adhering to the defined verification criteria. Document the outcomes of the static verification process.
- BP4: Perform testing on software units based on the unit test specification aligned with the software unit verification strategy. Document the test results and associated logs.
- BP5: Establish bidirectional traceability, connecting software units to static verification results. Establish bidirectional traceability between the software's detailed design and the unit test specification. Additionally, establish bidirectional traceability between the unit test specification and the corresponding unit test results.
- BP6: Maintain consistency by ensuring alignment between the software detailed design and the unit test specification.
- BP7: Summarize the results of unit tests and static verification, and effectively communicate these results to all relevant stakeholders.

Now that we have discussed the base practices related to software unit verification (SWE.4), let's delve into the key performance indicators associated with this component:

Code Coverage:

Code coverage, as mentioned in [22], measures the percentage of executed source code during a specific test suite. High test coverage indicates extensive code execution, reducing the likelihood of undiscovered software bugs compared to low coverage. Metrics such as subroutine and statement coverage percentages are used to assess test coverage. Achieving high code coverage ensures thorough testing and minimizes the risk of undetected issues in the software.

SWE.04	Code Coverage
Statement	90%
Branch	81%
MC/DC	67%

Table 3.14: Code coverage KPI

Statement coverage means whether each statement in the program has been executed. It aims to cover all the statements while executing a program at least once.

Branch coverage means whether each branch in the program has been covered with True and False. It focuses on covering both conditional and unconditional branches. It is a stronger criterion than the statement coverage as covering all branches also implies covering all statements [23].

Modified Condition/Decision Coverage (MC/DC) is used in software testing to test highly critical systems. This criterion mandates that all possible states of each condition should be tested while keeping other conditions fixed, ensuring thorough coverage. Additionally, MC/DC demands that a change in an individual condition should be demonstrated to have an impact on the final result.

Unit Test Specification Coverage (UTCov):

MC/DC, a testing method commonly employed for highly critical systems, ensures comprehensive coverage by testing all possible states of each condition while keeping other conditions constant. It goes a step further by requiring that a change in a single condition must affect the final result, thereby demonstrating the impact of individual condition changes. This criterion is crucial for thorough testing and validating the reliability of critical software systems.

SWE.04	UTCov								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
No. Of SWUD	555	600	675	675	800	825	825	830	832	832
SWUD traced (from test spec)	200	200	300	300	600	600	700	700	800	800
% Coverage	36%	33%	44%	44%	75%	73%	85%	84%	96%	96%

Table 3.15: Unit Test Specification Coverage KPI

Unit Test Result:

SWE.04	Unit Test Result									
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
Total Testcase	10	15	25	75	100	112	115	115	117	117
Run	0	0	0	25	30	40	75	100	117	117
Pass	0	0	0	25	30	38	70	100	110	117
% Pass	0%	0%	0%	33%	30%	34%	61%	87%	94%	100%

Table 3.16: Unit Test Result KPI

3.1.2.5 SOFTWARE INTEGRATION AND INTEGRATION TEST – SWE.5 in Automotive SPICE KPIs

The Software Integration and Integration Test process, referred to as SWE.5 in Automotive SPICE [24], assists organizations in effectively integrating individual software architecture elements. This process includes thorough testing to validate the proper functioning and interaction of these elements based on the specified software architecture. By following this process, organizations can guarantee a smooth integration of all software components, ensuring their cohesive operation as originally intended. To demonstrate compliance of the integrated software items with the software architectural design, the following best practices are recommended:

- BP1: Establish a software integration strategy that aligns with the project plan and release plan. This involves identifying the software items based on the software architectural design and defining a logical sequence for their integration.

- BP2: Formulate a software integration test strategy, including a regression test strategy, to validate the integrated software items following the established integration strategy. The regression test strategy ensures that integrated software items are retested if any software item undergoes changes.

- BP3: Develop a comprehensive test specification for the software integration test. This includes creating test cases based on the software integration test strategy for each integrated software item. The test specification should be designed to provide evidence of compliance of the integrated software items with the software architectural design.

- BP4: Carry out the integration of software units into software items and software items into the integrated software as outlined in the software integration strategy.

- BP5: Choose appropriate test cases from the software integration test specification. Ensure that the selected test cases provide adequate coverage based on the software integration test strategy and the release plan.

- BP6: Conduct the software integration test by executing the chosen test cases. Document the results and logs generated during the integration testing process

- BP7: Establish bidirectional traceability by linking the elements of the software architectural design with the test cases specified in the software integration test specification. Additionally, establish bidirectional traceability between the test cases included in the software integration test specification and the results obtained from the software integration tests. This traceability facilitates coverage analysis, consistency checks, and impact analysis.

- BP8: Ensure consistency by maintaining coherence between the elements of the software architectural design and the test cases outlined in the software integration test specification. Bidirectional traceability aids in ensuring this consistency, which can also be verified through review records.

- BP9: Summarize the results of the software integration tests and effectively communicate them to all relevant parties. This includes consolidating the test outcomes, analyzing the findings, and conveying the information to the stakeholders involved in the software integration process.

Now that we have discussed the base practices related to software unit verification (SWE.5), let's delve into the key performance indicators associated with this component:

Integration Test Specification Coverage (ITCov):

Through practical experience, we have come to understand that the isolation property of unit tests may not always suffice for certain functions. In such cases, one possible solution is to conduct tests that assess how different parts of the application function collectively as a cohesive whole. This approach is known as integration testing [25]. Unlike unit testing, integration testing takes into account the potential side effects right from the start. In fact, these side effects may even be intentional and desired outcomes of the testing process.

Integration test plays a crucial role in uncovering issues that may not be immediately apparent through individual examination of an application or specific unit. It focuses on identifying defects that arise from the interaction between different parts of the application. Oftentimes, these defects can be difficult to track or replicate, making integration testing all the more essential in ensuring the overall reliability and functionality of the system.

SWE.05	ITCov									
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
No. Of SWA	555	600	675	675	800	825	825	830	832	832
SWA traced (from test spec)	100	100	150	300	600	600	600	600	700	725
% Coverage	18%	17%	22%	44%	75%	73%	73%	72%	84%	87%

Table 3.17: Integration Test Specification Coverage KPI

Integration Test Result:

SWE.05	Int. Test Result									
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
Total Testcase	10	15	25	75	100	112	115	115	117	117
Run	0	0	0	25	30	40	75	100	117	117
Pass	0	0	0	5	5	5	40	100	100	117
% Pass	0%	0%	0%	7%	5%	4%	35%	87%	85%	100%

Table 3.18: Integration test KPI

3.1.2.6 SOFTWARE QUALIFICATION TEST – SWE.6 in Automotive SPICE KPIs

The Software Qualification Test process, part of Automotive SPICE SWE.6, plays a vital role in ensuring that the integrated software aligns with the defined software requirements. The purpose of this process is to assess the software against the established requirements and verify if they have been fully met and correctly implemented. As the Software Qualification Test is conducted shortly before software delivery, it closely relates to other processes such as Project Management, Configuration Management, Product Release, and Software Requirements Analysis. A successful Software Qualification Test is crucial as it helps identify potential errors that might otherwise go unnoticed, thereby maintaining customer satisfaction. The test environment employed can vary based on the product, with examples including SIL, PIL, or HIL.

Software Test Result:

SWE.06	ST Test Result								
	Wk1	Wk2	Wk3	Wk4	Wk5	Wk6	Wk7	Wk8	Wk9	Wk10
Total Testcase	10	15	25	75	100	112	115	115	117	117
Run	0	0	0	25	30	40	75	100	117	117
Pass	0	0	0	25	30	40	75	90	90	117
% Pass	0%	0%	0%	33%	30%	36%	65%	78%	77%	100%

Table 3.19: Software Tester test results KPI

3.2 KPI data retrieval

In several applications spanning diverse industries, data retrieval is essential. Organizations rely on fast and accurate information in today's data-driven environment to make wise decisions, get insights, and foster corporate growth. Accessing current and relevant data is essential for constructing intelligent systems as well as for market research, competitive analysis, and data analysis.

Data retrieval plays a vital role in the development of dashboards, especially when the aim is to present real-time data to users. Dashboards serve as a visual representation of critical information, allowing users to monitor and analyze key metrics and make informed decisions. To achieve this, the process of data retrieval becomes crucial. Real-time data retrieval ensures that the information displayed on the dashboard is constantly updated, providing users with the most current insights on the software process development.

To ensure successful data retrieval for dashboard development, considerations such as data source compatibility, data validation and cleansing, and efficient querying techniques need to be addressed. Additionally, attention should be given to data security and privacy, as sensitive information may be involved.

The majority of the KPI data mentioned in the previous section is sourced from web pages, which necessitates the use of web scraping techniques for data extraction. Additionally, a portion of the data is obtained from CSV and XLS files, employing file-based retrieval methods. The remaining data is fetched from online platforms, through dedicated Application programming interface (API) channels.

3.2.1 Data retrieval using web scrapping

Web scraping, also referred to as web extraction or harvesting, is a method used to extract data from the World Wide Web (WWW) and store it in a file system or database for future retrieval or analysis [26]. The process of obtaining data from the internet can be broken down into two consecutive steps: acquiring web resources and extracting specific information from the acquired data.

3.2.1.1 Acquiring web resources

The process of web scraping begins with a web scraping program sending an HTTP request to a targeted website in order to acquire resources. This request can be in the form of a URL with a GET query or a section of an HTTP message with a POST query.

A GET request is one method of communication used for web interactions. It is a request made by a client to retrieve a specific resource from a server. After processing the request, the server responds by providing the requested resource.

On the other hand, a POST request is utilized by a client to submit data that needs to be processed by a server. Unlike a GET request, which retrieves data, a POST request is used to send data to the server for the purpose of creating, updating, or modifying a resource (refer to Figure 3.7 for more details).

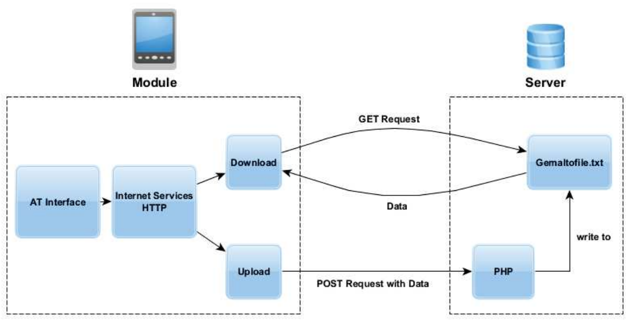


Figure 3.7: POST and GET requests

The moment the targeted website successfully receives and processes the request, the demanded resource is retrieved and sent back to the web scraping program. The resource can be received in various formats, such as HTML-based web pages, XML or JSON data feeds, or multimedia files like images, audio, or videos. After downloading the web data, the extraction process proceeds to parse, reformat, and organize the data in a structured manner.

The main modules used to compose HTTP requests are Urllib2 or Selenium, The Urllib2 module offers a range of functions for handling HTTP requests, including authentication, redirections, and cookies. On the other hand, Selenium acts as a web browser wrapper that allows users to automate website browsing by programming, utilizing browsers like Google Chrome or Internet Explorer.

3.2.1.2 Data parsing and extraction

After getting the requested data, we move to the process of extracting specific information from the HTML markup and converting it into a structured format that can be easily processed and understood by a computer program. The HTML file is parsed by an HTML parser, which analyzes the structure and elements of the HTML document. This process involves breaking down the HTML code into a structured representation known as a parse tree or DOM (Document Object Model).

For data extraction, Beautiful Soup is specifically designed for scraping HTML and XML documents. It provides convenient Pythonic functions for navigating, searching, and modifying a parse tree, check Figure 3.8. Beautiful Soup also offers a toolkit for decomposing HTML files and extracting desired information using libraries like lxml or html5lib. Additionally, Beautiful Soup can automatically detect the encoding of the parsed content and convert it to a format readable by the client. Similarly, Pyquery provides a set of Jquery-like functions for parsing XML documents. However, unlike Beautiful Soup, Pyquery exclusively supports lxml for efficient XML processing.

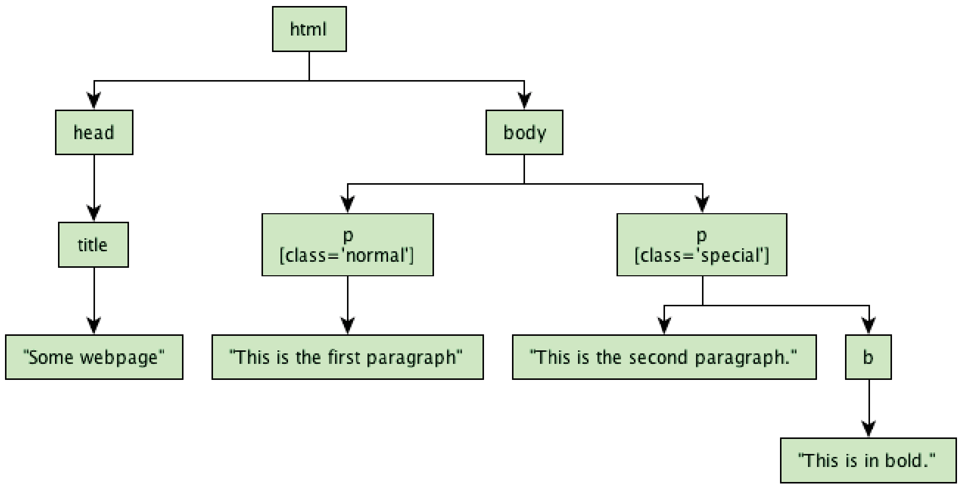


Figure 3.8: HTML flow chart [27]

To fetch the data from Continuous integration tools and some platforms like Reqtify and Polyspace Bug Finder or Code Prover tools, we used web scraping and Python libraries. Most web pages are structured in tables, for this reason, we created a Python class that transforms the HTML page into data frames. The class has different functions as shown in the algorithm below:

Algorithm 1: HTML to DataFrame Conversion

```
1  Static Method get_html_tables(html: str) → list[list[str]]
       Input  : HTML content as a string
       Output: List of HTML tables as a nested list of strings
2      if html then
3          soup ← BeautifulSoup(html, ”html.parser”)
4          soup.prettify()
5          extractTables ← soup.find_all(”table”)
6          return extractTables
7      end
8  Static Method select_table(tag: str, tableFeature: str, allTables: list, all:
     int) → list[str]
       Input  : Tag name as a string, table feature as a string, list of tables, flag for
                returning all tables
       Output: Specific table(s) as a list of strings
9      try  htmlTable ← [table for table in allTables for header in
        table.find_all(tag) if tableFeature == header.text]
10     if all = 0 then
11         return htmlTable[0]
12     end
13     else
14         return htmlTable
15     end
16     catch TypeError, IndexError  return None
17 Static Method get_value(dfTable, row: str, column: str) → str
       Input  : Dataframe table, row name as a string, column name as a string
       Output: Value from the specified row and column as a string
18     dfValue ← dfTable[dfTable.values == row][column].item()
19     return dfValue
20 Static Method list_2df(table) → pd.DataFrame
       Input  : HTML table as a list of strings
       Output: Dataframe representation of the table
21     ex ← Extractor(table)
22     ex.parse()
23     listOfLines ← ex.returnList()
24     df ← pd.DataFrame(listOfLines[1:], columns=listOfLines[0])
25     return df
```

3.2.2 Application Programming Interfaces (APIs)

3.2.2.1 What's an API?

An application programming interface (API) enables communication between two or more computer programs. It serves as a software interface that enables your product or service to provide services to other products or services [28]. In simple words, APIs are messengers that take requests and tell the system what is requested to be done and then return the response back to the source.

3.2.2.2 How do APIs work?

One of the most familiar examples of an API is to think of it as a waiter in a restaurant. Imagine some hosts sitting at a table with a menu of choices to order from. And the kitchen is the part of the system which will prepare the order. So the waiter is the critical link to communicating your order to the kitchen, in an easy-to-follow manner and delivering the food back to the table. From their handwritten notes to the computer system to the kitchen cooks, they translate your request for pancakes, and ultimately they come back with your small stack. And that's what an API (or in this case the waiter) does, view Figure 3.9.

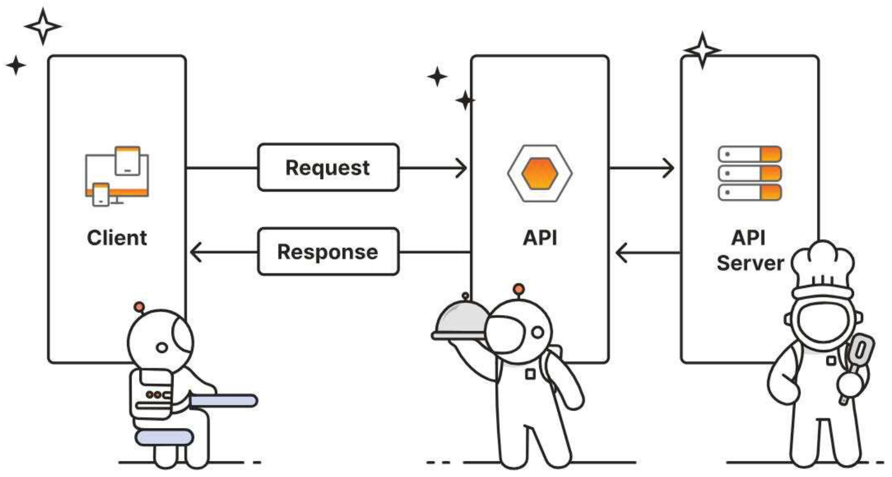


Figure 3.9: How do APIs work? [29]

There are four types of API actions:

- GET: This action is to request data from a server.
- POST: This action is to send new information to a server.
- PUT: This action is to make changes to existing data on a server.
- DELETE: This action is to remove already existing data from a server.

3.2.2.3 Why APIs are important ?

APIs are of paramount importance in modern software development. They facilitate seamless communication and data sharing between different software systems and applications. By providing pre-built functions and modules, APIs promote code reusability and accelerate the development process. They enable modular development, allowing developers to create independent software components that can be easily connected via

APIs, enhancing scalability and modularity. APIs also foster collaboration and innovation by encouraging third-party developers to build applications and services that integrate with existing platforms. In the mobile and web development realm, APIs provide a standardized way to access data and functionality from servers, enabling the creation of interactive and dynamic applications. Additionally, APIs are essential in service-oriented architectures, enabling the exposure and utilization of capabilities in a standardized and interoperable manner. Overall, APIs simplify development, promote integration, foster collaboration, and enhance the capabilities of software applications and services in the modern digital landscape.

3.2.2.4 SOAP vs. REST APIs

SOAP (Simple Object Access technology) API provides organized data exchange between computer systems. Requests are sent in XML format, and replies are sent in XML format as well. Interoperability between different platforms and programming languages is a key feature of SOAP APIs. They include cutting-edge features including error management, support for intricate computations, and encryption. Even though SOAP API implementation might be complicated, they are frequently utilized in business settings that value security and resilience.

REST (Representational state transfer) was developed in order to offer a more user-friendly method of accessing internet services. Modern web-based application development commonly takes advantage of the architectural pattern known as REST [30]. Depending on how it is created, what is added to it, and the use it is intended for, a REST API might be very simple or very complex. When resources are limited, rigorous security is not necessary, browser client compatibility is vital, and data integrity and scalability are needed, they are acceptable.

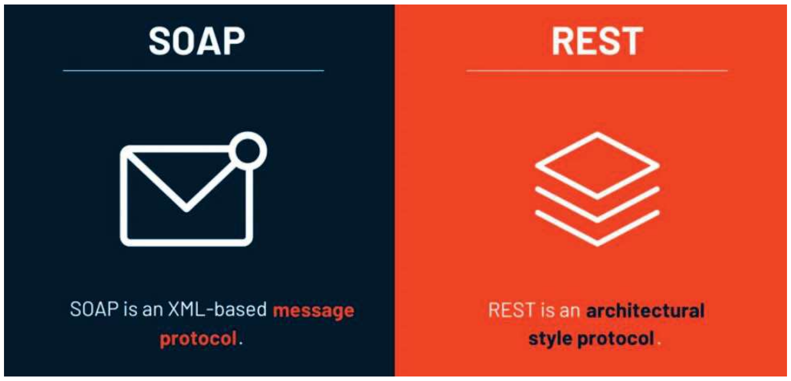

SOAP
REST
SOAP is an XML-based message protocol.
REST is an architectural style protocol.

Figure 3.10: REST Vs SOAP APIs [31]

3.2.2.5 APIs for fetching data

In order to retrieve data related to JIRA KPIs, REST APIs provided by JIRA were utilized. These APIs allow seamless communication with the JIRA system, enabling the extraction of specific information through tailored queries. By leveraging these REST

APIs, we were able to define and execute requests to fetch the desired KPI data from JIRA. This approach provided a standardized and efficient method for obtaining the necessary metrics and insights, empowering us to analyze and monitor the performance of our projects and workflows within the JIRA ecosystem.

In addition to the REST APIs, JIRA also offers a range of filters that enhance the data retrieval process as shown in Figure 3.11. These filters provide an intuitive and flexible way to select specific data based on various criteria. By utilizing JIRA's filter functionality, we were able to refine our data retrieval process and narrow down the information to meet our specific requirements. These filters allowed us to focus on relevant project data, such as specific issue types, statuses, priorities, or timeframes. By leveraging these powerful filtering capabilities, we could efficiently extract and analyze the data that was most pertinent to our KPI tracking and reporting needs.

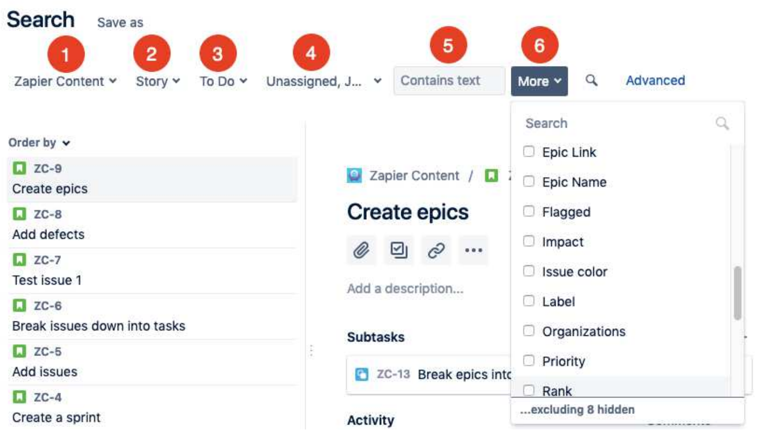


Figure 3.11: How to create a JIRA filter

Some of the queries that were used to fetch the needed data from JIRA are:

```
wp_postccb_not_analysed = project in ("183982309", "8781637293",
   ↪ "623872735", "76273698")
AND issuetype in (ChangeRequest, ProblemReport)
AND status in (Closed, "Accepted Assigned", Started, Verified,
   ↪ Deferred, Rejected, Closed_FixFuture, Closed_Done, "Transfer to
   ↪  CR", Implemented)

wp_defects = project in ("387674903", "903898747", "397984687",
   ↪ "287633094") AND issuetype in (ChangeRequest, ProblemReport)
   ↪ AND "Issue Category" = Bug}

wp_open = project in ("093094874", "309984784", "124823480",
   ↪ "0408900943") AND issuetype in (ChangeRequest, ProblemReport)
   ↪ AND status in (Submitted, "In Analysis", "Accepted Assigned",
   ↪ Started, Verified, Deferred, "In Rework", Implemented) AND "
   ↪ Issue Category" = Bug

wp_cr = project in ("908907234", "0937874094", "983749827",
   ↪ "988742849") AND issuetype = ChangeRequest
```

```
wp_pr = project in ("904982743", "934878374", "0000069640",
   ↪ "0000069641") AND issuetype = ProblemReport

wp_preccb_not_analysed = project in ("734398232", "98786474",
   ↪ "7687634723", "876468734") AND issuetype in (ChangeRequest,
   ↪ ProblemReport) AND status in (Submitted, "In Analysis", "In
   ↪ Rework")

wp_postccb_not_analysed = project in ("8746764387", "8736487634",
   ↪ "008374829", "87837498") AND issuetype in (ChangeRequest,
   ↪ ProblemReport) AND status in (Closed, "Accepted Assigned",
   ↪ Started, Verified, Deferred, Rejected, Closed_FixFuture,
   ↪ Closed_Done, "Transfer to CR", Implemented)
```

3.2.3 File-based Retrieval

File-based data retrieval refers to the process of accessing and extracting data from files stored on a computer or a file system. This approach involves reading and parsing data from various types of files, such as CSV (Comma-Separated Values), Excel spreadsheets (XLS or XLSX), JSON (JavaScript Object Notation), XML (eXtensible Markup Language), and others.

In file-based data retrieval, the data is typically structured in a specific format within the files. The retrieval process involves opening the file, reading its contents, and extracting the relevant data elements. This can be done using programming languages or tools that provide file input/output operations, parsing capabilities, and data manipulation functions like Python.

File-based data retrieval is commonly used when working with locally stored data files or when exchanging data with external systems that provide data in file formats. It offers a flexible and convenient method for accessing and processing structured data stored in files.

In the context of fetching KPI's data, it is worth noting that the data required for analysis and reporting can be sourced from various systems and formats. In some cases, KPI data resides within continuous integration tools, where metrics and measurements are collected during the execution of software pipelines. Additionally, reports containing valuable insights are generated in formats such as XLS (Excel) and XML (eXtensible Markup Language) after the completion of pipeline runs. To retrieve and gather the necessary information, we employ a file-based retrieval approach. This method allows us to access and extract the relevant data from these generated files, enabling us to incorporate it into our KPI analysis and reporting processes.

Conclusion

In this chapter, we leveraged the key performance indicators (KPIs) necessary for our dashboard and explored various methods for data retrieval. We discussed the importance of data retrieval in extracting real-time information and uncovering insights and the methods used. These methods provide us with the necessary data to populate our dashboards and make informed decisions. By employing efficient data retrieval techniques, we ensure the availability of up-to-date and accurate information, enabling us to monitor and analyze KPIs effectively.

Chapter 4

Data Collection and Database Design

Introduction

In this chapter, we delve into the crucial process of creating an efficient and well-structured database for our dashboard. We explore the principles and methodologies involved in designing a robust database schema that ensures data integrity and supports optimal data storage and retrieval.

4.1 Introduction to databases

4.1.1 What is a database?

A database is a structured and organized collection of information or data that is stored electronically in a computer system. It is managed by a database management system (DBMS), which controls and handles the data. The combination of the data, DBMS, and associated applications is referred to as a database system. In modern databases, data is typically arranged in tables with rows and columns, allowing for efficient processing and querying. This enables easy access, management, modification, update, control, and organization of the data [32]. The common language used for writing and querying data in most databases is structured query language (SQL).

4.1.2 What is a DBMS?

Database Management Systems (DBMS) are software systems designed to store, retrieve, and perform queries on data. They act as an intermediary between users and databases, enabling users to create, read, update, and delete data within the database [33]. A DBMS takes care of managing the data itself, the database engine responsible for processing operations, and the database schema that defines the structure and organization of the data. This allows users and other programs to manipulate and extract data while ensuring data security, integrity, concurrency, and consistent data administration procedures, check Figure 4.1.

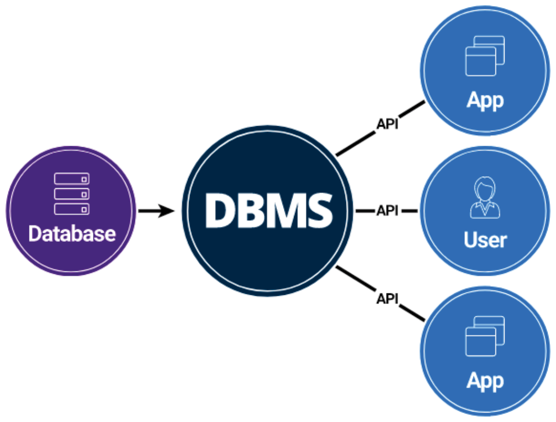


Figure 4.1: Database management system

4.1.3 What are the types of Databases?

There are various types of databases, each designed to cater to specific requirements and manage different types of data. Here are some commonly encountered types of databases:

- Relational Databases (RDBMS) [34]: Relational databases organize data into tables with rows and columns, using SQL to manipulate and query the data. They offer a structured and efficient approach to storing and retrieving structured data.
- NoSQL Database [35]s: NoSQL databases are non-relational databases that provide flexibility in handling unstructured or rapidly changing data. They are particularly suitable for scalable and distributed architectures where quick data retrieval is essential.
- Object-Oriented Databases (OODBMS) [36]: Object-oriented databases store data in objects, incorporating both data and associated behaviors. They are well-suited for applications that require complex data structures and leverage object-oriented programming concepts.
- Hierarchical Databases [37]: Hierarchical databases organize data in a tree-like structure, establishing parent-child relationships between data elements. They are suitable for managing data with inherent hierarchical characteristics, such as file systems or organizational structures.
- Network Databases: Network databases are similar to hierarchical databases but allow more intricate relationships between data elements. They employ a network model to represent data, enabling multiple records to be linked together.
- Graph Databases [38]: Graph databases utilize graph structures to store and manage data. They excel in scenarios involving highly interconnected data and prove valuable in managing complex relationships and performing graph-based algorithms.
- Time-Series Databases [39]: Time-series databases specialize in storing and analyzing time-stamped data, such as sensor readings or financial data. They efficiently handle data points over time and support specific time-based queries and aggregations.

These are just a few examples of the different types of databases available, each tailored to meet specific data management needs. The selection of a database type depends on factors like data characteristics, application requirements, scalability demands, and performance considerations.

4.1.4 What is Structured Query Language (SQL)

SQL is a computer language, which is flexible and crucial for handling relational databases. Users may interact with databases using this tool to complete a variety of activities, including data retrieval, updating, and organization. A standardized set of

SQL commands and syntax enables smooth communication between users and databases, enabling sophisticated data manipulation and complicated procedures. It gives users the ability to build complex queries, apply data integrity policies, and carry out in-depth computations and analyses, Check Figure 4.2. SQL is essential for effectively dealing with structured data because of its wide acceptance and interoperability with many database management systems.

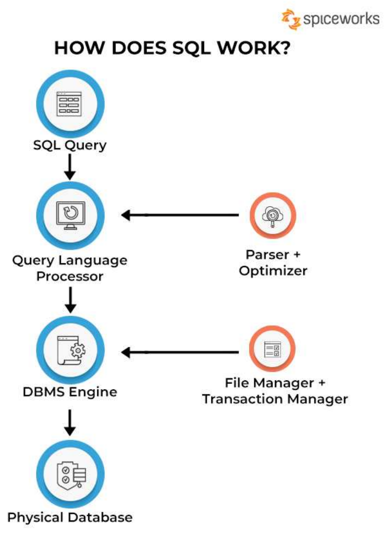


Figure 4.2: How does SQL work? [39]

4.2 Designing and implementing a database for the ASPICE dashboard

4.2.1 Database requirement analysis

4.2.1.1 Objective of the ASPICE database

The objective of designing an ASPICE database is to provide an efficient and organized storage solution for the dashboard data and update information related to ASPICE. By structuring the data in a database, it becomes easier to manage and retrieve the required information for the dashboard. The database allows for the seamless updating of data related to ASPICE, ensuring that the dashboard reflects the most recent and accurate information. Additionally, a structured database enables the implementation of data integrity rules, data validation, and efficient querying, promoting data consistency and reliability. The ultimate goal is to create a reliable and scalable database system that supports the effective management and updating of dashboard data in line with ASPICE requirements.

We opted to implement a relational database for the ASPICE dashboard based on several factors. Firstly, relational databases excel at organizing structured data into tables with defined relationships, allowing for efficient data management. This structure ensures the integrity of the information presented on the dashboard. Additionally, the versatility of SQL queries supported by relational databases enables me to extract valuable insights through complex data retrieval and analysis. The scalability of relational databases was also a key consideration, as they can handle large data volumes and accommodate future growth. Lastly, the extensive ecosystem surrounding relational databases provides ample resources and support for managing, modeling, and integrating the database into my dashboard application. In summary, the decision to utilize a relational database was driven by its strength in structured data management, query flexibility, scalability, and the availability of a robust ecosystem.

4.2.1.2 Users and stakeholders of the ASPICE dashboard

The users of an ASPICE dashboard can vary depending on the context and purpose of the dashboard. Typically, the primary users of an ASPICE dashboard are individuals and teams involved in software development and engineering processes within an organization. These users can be:

- Team manager: they utilize the dashboard to monitor team performance, track progress, and identify areas for improvement.
- Project managers: they rely on the dashboard to gain visibility into project status, resource allocation, and adherence to ASPICE requirements. They can make informed decisions, manage project risks, and ensure successful project outcomes.
- Quality assurance teams: they rely on the dashboard to monitor and assess the quality of software development processes and ensure compliance with ASPICE standards. They can track metrics, identify potential issues, and take corrective actions to improve quality throughout the software development lifecycle.
- Software engineers: they maintain and ensure the functionality of the ASPICE dashboard. They address technical issues, implement updates, optimize performance, and enhance security. Testing and documentation are carried out to ensure reliability, and backup mechanisms are established for data protection. Their role is vital in providing a well-functioning and reliable dashboard for users.

The ASPICE dashboard serves as a centralized platform for these users to access and visualize key metrics, KPIs, and process-related information related to software development projects following the Automotive SPICE framework. It helps facilitate decision-making, performance tracking, and process improvement efforts by providing relevant and real-time insights to the users.

4.2.1.3 Functional requirements

The database has to store all the chosen data related to ASPICE compliance metrics in every site of Infineon and for every single ongoing project in each department, to correctly visualize and interpret them in the ASPICE dashboard system. In particular, the system must efficiently store:

- The ongoing project's data, with their identifiers.
- The team data like the name and the site identifiers.
- The KPI's data.

The database should manage login operations from users and admin, to provide different rights for each user role. And it must allow the administrator to define, configure, and monitor roles in the application.

The system must allow the maintenance engineer with admin credentials to :

- View and edit KPI's information.
- Insert and define new metrics.
- Alter unused historical data.

4.2.1.4 Non functional requirements

Non-functional requirements are criteria that describe the qualities, characteristics, and constraints of a system or software application, rather than its specific functionality. Unlike functional requirements that focus on what the system should do, non-functional requirements define how the system should perform and behave in terms of its performance, usability, reliability, security, and other important aspects.

The database should encompass:

- Performance: Verify the dashboard's response time, throughput, and resource usage requirements, ensuring it meets performance expectations under various conditions and user loads.
- Reliability: Check the database's ability to perform consistently and reliably over time, including measures such as availability, fault tolerance, and error handling.
- Security: Specifies the system's security measures and requirements, including access control, data protection, authentication, and encryption to ensure the confidentiality, integrity, and availability of sensitive information.
- Scalability: Defines the database's ability to handle the increased workload or accommodate growth in terms of users, data volume, or system components, ensuring it can scale effectively without significant performance degradation.
- Maintainability: Describes the ease with which the system can be modified, updated, repaired, and extended over its lifecycle.

- Compliance: Specifies any legal, regulatory, or industry-specific standards and requirements that the system must adhere to, ensuring it meets relevant compliance obligations.

4.2.1.5 Constraints

The DBMS application should satisfy the following additional constraints:

- Be implemented with MySQL.
- High security as it contains internal data, protecting sensitive data by defining access controls, user privileges, and data encryption methods.
- Database optimization by using some techniques to enhance its performance, such as indexing, partitioning, and query optimization strategies.

4.2.2 Database conceptual and logical design

The main goal of the conceptual design phase is to build a conceptual model based on the previously specified criteria and get it closer to the final physical model. An entity-relationship model is a type of conceptual model that is often utilized [40].

4.2.2.1 Entity-Relationship Schema

An entity-relationship (ER) schema [41] is a graphical representation of the database's entities, properties, and connections. It works as a visual tool for designing and outlining a database system's structure. A clear understanding of the organization of the database is made possible by ER diagrams, which describe entities, properties, and connections using symbols and shapes.

One may learn more about the entities that are present in the database, their related properties, and the links between them by utilizing an ER diagram. The graphic gives a clear summary of the database schema, which makes it simpler to understand the connections between elements. ER diagrams are frequently used in database design to improve database creation, ease maintenance of the database system, and facilitate communication.
The relationships between the ASPICE database tables can be described as follows:

- Sites Table and Projects Table: This relationship is a one-to-many relationship, where each site can have multiple projects, but each project belongs to only one site. This is represented by the foreign key relationship between the site ID in the Sites table and the site ID in the Projects table.
- Projects Table and KPI Tables: This relationship is a one-to-one relationship, where each project can have only one KPI, and each KPI belongs to only one project. This is represented by the foreign key relationship between the project ID in the Projects table and the project ID in each KPI table.

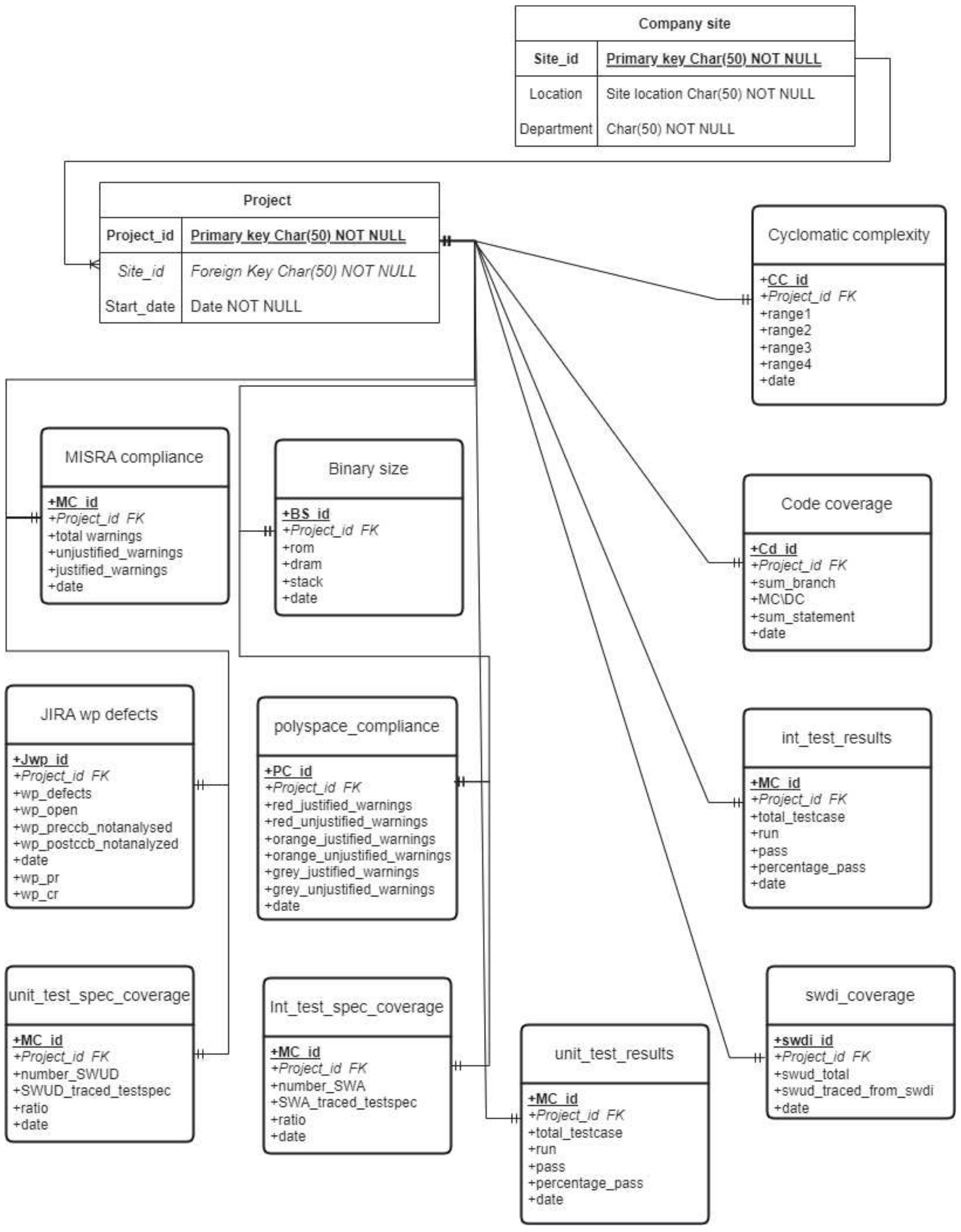


Figure 4.3: ER schema of ASPICE database

4.2.2.2 Data dictionary

Entity	Description	Attributes	Identifier
Company site	This table represents the set of sites that the company has all over the world	site_id location department	site_id
Project	This table represents the set of ongoing projects	project_id site_id start_date	project_id
Cyclomatic Complexity	This table represents the cyclomatic complexity metric, and the number of functions having a CC in a certain range	CC_id project_id range1 range2 range3 range4 date	CC_id
MISRA compliance	This table represents the Misra compliance KPI	MC_id project_id total warnings unjustified warnings justified warnings date	MC_id
Binary size	This table represents the binary size KPI	BS_id project_id rom dram stack date	BS_id
Code coverage	This table represents the code coverage KPI	Cd_id project_id sum_branch MC\DC sum_statement date	Cd_id
Jira wp defects	This table represents the Jira work product defects	Jwp_id project_id wp_defect wp_open wp_preccb_notanalyzed wp_postccb_notanalyzed wp_pr wp_cr date	jwp_id

Polyspace compliance	This table represents the polyspace compliance KPI	PC_id project_id red_justified_warnings red_unjustified_warnings orange_justified_warnings orange_unjustified_warning grey_justified_warnings grey_unjustified_warnings date	PC_id
Int_test_results	This table represents the integration test results	int_id Project_id total_testcase run pass percentage_pass date	int_id
Unit_test_results	This table represents the unit test results	uni_id Project_id total_testcase run pass percentage_pass date	uni_id
Int_test_spec _coverage	This table represents the integration test specification coverage	Int_spec_id Project_id number_SWA SWA_traced_testspec ratio date	int_spec_id
Unit_test_spec _coverage	This table represents the unit test specification coverage	Unit_spec_id Project_id number_SWUD SWUD_traced_testspec ratio date	unit_spec_id

Table 4.1: ASPICE database dictionary

4.2.3 Database physical design

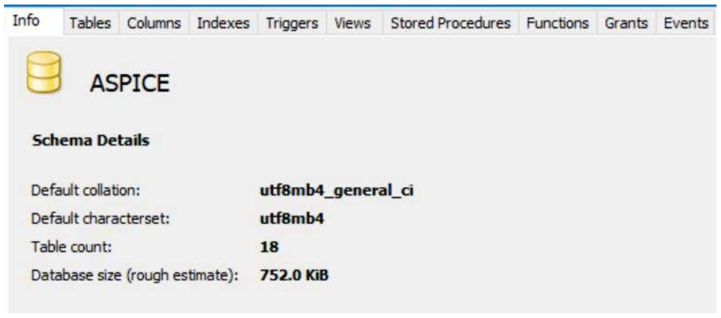
A physical schema, also known as a physical data model, is a representation of the database structure at the physical level. It describes how the data is physically stored and organized in the database system. The physical schema includes details such as the storage format, indexing mechanisms, partitioning strategies, and any other physical implementation considerations.

The physical schema is derived from the logical schema, which defines the database structure at a conceptual level, and it takes into account the specific characteristics and constraints of the database management system and the underlying hardware infrastructure. Physical schema optimization aims to achieve efficient data storage, retrieval, and performance by considering factors such as disk space utilization, data access patterns, and query optimization techniques. In short, it specifies the actual implementation of the database design, taking into consideration the technical aspects of storage and performance to ensure the efficient management of data in the physical database environment.

4.2.3.1 Database creation

```
CREATE DATABASE `ASPICE`
  CHARACTER SET utf8
  COLLATE utf8_general_ci;
```

The CHARACTER SET utf8 specifies the encoding type as UTF-8, which supports a wide range of characters from different languages. The COLLATE utf8_general_ci specifies the collation for the character set, which determines the rules for comparing and sorting the data check Figure 4.4.



ASPICE

Schema Details

Default collation:	utf8mb4_general_ci
Default characterset:	utf8mb4
Table count:	18
Database size (rough estimate):	752.0 KiB

Figure 4.4: ASPICE database Info

4.2.3.2 Table creation

Create the tables in the right order with respect to foreign keys

```
CREATE TABLE `site` (
  `site_id` VARCHAR(50) PRIMARY KEY NOT NULL,
  `location` VARCHAR(50) NOT NULL,
  `department` VARCHAR(50) NOT NULL,
);

CREATE TABLE `project` (
  `project_id` VARCHAR(50) PRIMARY KEY NOT NULL,
  `start_date` DATE NOT NULL,
  `site_id` VARCHAR(50),

  FOREIGN KEY (`site_id`) REFERENCES `site` (`site_id`)
);

CREATE TABLE `cyclomatic_complexity` (
  `cc_id` BIGINT(20) PRIMARY KEY NOT NULL AUTOINCREMENT,
  `project_id` VARCHAR(50) NOT NULL,
  `range1` VARCHAR(50) NOT NULL,
  `range2` VARCHAR(50) NOT NULL,
  `range3` VARCHAR(50) NOT NULL,
  `range4` VARCHAR(50) NOT NULL,
  `date` DATE NOT NULL,

  FOREIGN KEY (`project_id`) REFERENCES `project` (`project_id`)
);

CREATE TABLE `misra_compliance` (
  `mc_id` int PRIMARY KEY NOT NULL AUTOINCREMENT,
  `project_id` VARCHAR(50) NOT NULL,
  `total_warnings` BIGINT(20) NOT NULL,
  `unjustified_warnings` BIGINT(20) NOT NULL,
  `justified_warnings` BIGINT(20) NOT NULL,
  `date` DATE NOT NULL,

  FOREIGN KEY (`project_id`) REFERENCES `project` (`project_id`)
);

CREATE TABLE `binary_size` (
  `bs_id` BIGINT(20) PRIMARY KEY NOT NULL AUTOINCREMENT,
  `project_id` VARCHAR(50) NOT NULL,
  `rom` FLOAT NOT NULL,
  `dram` FLOAT NOT NULL,
  `stack` FLOAT NOT NULL,
  `date` DATE NOT NULL,

  FOREIGN KEY (`project_id`) REFERENCES `project` (`project_id`)
);

CREATE TABLE `binary_size` (
  `bs_id` BIGINT(20) PRIMARY KEY NOT NULL AUTOINCREMENT,
  `project_id` VARCHAR(50) NOT NULL,
  `rom` FLOAT NOT NULL,
  `dram` FLOAT NOT NULL,
  `stack` FLOAT NOT NULL,
  `date` DATE NOT NULL,

  FOREIGN KEY (`project_id`) REFERENCES `project` (`project_id`)
);

CREATE TABLE `code_coverage` (
  `cd_id` BIGINT(20) PRIMARY KEY NOT NULL AUTOINCREMENT,
  `project_id` VARCHAR(50) NOT NULL,
  `sum_branch` FLOAT NOT NULL,
  `MC\DC` FLOAT NOT NULL,
  `sum_statement` FLOAT NOT NULL,
  `date` DATE NOT NULL,

  FOREIGN KEY (`project_id`) REFERENCES `project` (`project_id`)
);
.
.
.
etc
```

4.2.3.3 Populate the database

The process of populating the database was automated using a Python script that utilized the SQLalchemy library. After extracting the relevant data from various tools, such as continuous integration systems or test management software, the Python script processed and transformed the data into a suitable format for insertion into the database. SQLalchemy, a powerful Python library for interacting with databases, was employed to establish a connection with the database and execute the necessary SQL statements for data insertion. By leveraging this approach, the database population process was streamlined and executed programmatically, ensuring efficiency and accuracy in transferring the extracted data into the database.

To facilitate the database population process, we created a Python class for each table in the database. These classes served as representations of the tables and provided a convenient way to interact with the database. We established a connection to the database using SQLalchemy, which allowed us to establish a communication channel between our Python script and the database.

Once the connection was established, we fetched the relevant data from various sources, such as external files or APIs, using appropriate data retrieval techniques. We then transformed the fetched data into suitable formats that matched the structure of the corresponding tables.

Using the Python classes we created, we performed the necessary operations to insert the fetched data into the respective tables of the database. This involved constructing SQL statements and executing them through the SQLalchemy library, ensuring that the data was accurately inserted into the designated tables.

By encapsulating the database operations within Python classes and leveraging SQLalchemy's functionality, we streamlined the process of populating the database. This approach provided a structured and efficient means of connecting to the database, fetching the required data, and inserting it into the appropriate tables, ensuring the integrity and consistency of the database contents.

4.2.3.4 Queriying the database

Query the binary_size data associated with project glomanhaV2_0_0 in Italy Padova.

```
SELECT *
FROM binary_size
WHERE project_name = 'glomanhaV2_0_0'
  AND site_id = 'ITPD';
```

Query the number of change requests from '2023-05-12' to '2023-06-12' associated with project 'glomanhaV2_0_0' in Austria Graz.

```
SELECT COUNT(wp_cr)
FROM Jira_wp_defects
WHERE date >= '2023-05-12' AND date <= '2023-06-12'
  AND project_id = 'glomanhaV2_0_0'
  AND site_id = 'AUGR'
```

Conclusion

In conclusion, in this chapter, we presented the database design for the ASPICE dashboard which provides a structured and efficient storage solution for managing the data related to KPIs, projects, and other relevant information. Moreover, by leveraging Database Management Systems (DBMS), the dashboard can effectively manage KPI's structured data and support real-time updates and streamlined data processing capabilities.

Chapter 5

Dashboard development and automation

Introduction

In this chapter, we delve into the key aspects of building an effective and user-friendly dashboard showcasing various charts, graphs, and visual elements that enhance the presentation of information. We begin with an overview of the chosen dashboard framework, highlighting its features and capabilities. We also focus on the integration of data fetching and database functionalities, explaining how we seamlessly connect to the underlying database to retrieve and update relevant data.

5.1 Overview of the dashboard framework

There are various options for dashboard development frameworks, each with its own strengths and considerations. Some popular frameworks include:

- Tableau: is a business intelligence and data visualization platform that allows users to connect, display, and exchange data in a very dynamic way. With a drag-and-drop interface, it enables users to swiftly examine and explore huge and complicated datasets without the need for coding or programming knowledge. Line charts, bar charts, maps, scatter plots, and many more chart kinds and visualization choices are available in Tableau [42].

- Power BI: Microsoft's business analytics service that enables users to visualize and analyze data with interactive dashboards, reports, and visualizations. It offers strong integration with Microsoft products and cloud services.

- QlikView/QlikSense: is a first-generation analytics platform. Qlik's data discovery and visualization tools allow users to create dashboards and reports in an interactive way. They offer powerful data exploration and associative search features.

- D3.js: This a JavaScript toolkit designed to build interactive and dynamic data visualizations exclusively for web browsers. It offers significant customization and control over the visualization design.

- Google Data Studio: A free online tool for creating customized data and dashboards, introduced by Google. This tool provides smooth integration with regard to various data sources.

5.1.1 Tableau for ASPICE dashboard

During the evaluation process, we assessed the functionality provided by each framework, including data integration capabilities, visualization options, interactive features, and Ease of use for both developers and end-users. We looked for a framework with user-friendly interfaces and intuitive tools. Additionally, it is important to examine the framework's ability to connect to various data sources and to handle scalability and performance requirements.

After careful consideration of various available tools, we made the decision to work with Tableau due to its numerous advantages. Tableau stood out as the most suitable choice based on factors such as powerful data visualization capabilities, a user-friendly interface, robust data integration features, interactive and dynamic functionality, scalability and performance, and a strong community support system. Taking all these factors into account, Tableau emerged as the optimal framework for our dashboard development, providing us with the necessary tools and resources to create a visually appealing and effective dashboard solution.

5.1.1.1 Data Source Connection

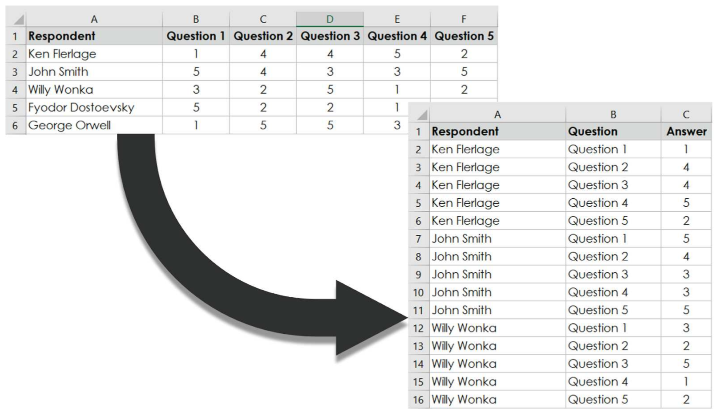
To create the KPI's graph in Tableau, it is essential to establish a connection with the ASPICE database. This connection allows Tableau to access and retrieve the relevant data needed for the graph. By connecting to the ASPICE database, Tableau can execute queries and fetch the necessary information, related to the KPIs, project details, and other relevant data points. This seamless integration between Tableau and the ASPICE database ensures that the graph is based on accurate and up-to-date data, enabling meaningful visualizations and insightful analysis.

To connect to the ASPICE database, we provided the required connection details, including the server name, port number, database name, and authentication credentials. Tableau attempted to connect to the specified data source using the information provided. If the connection was successful, we could proceed with data exploration and visualization. Once connected, we selected the tables and views from the ASPICE database we wanted to work with and define any necessary links or filters. Tableau imported the data into the workbook and allowed us to create visualizations, perform analysis and create the dashboard based on the connected data source.

5.1.1.2 Data pivoting and filtering using Tableau

Data pivoting is a technique used to transform data from a row-based format to a column-based format. Essentially, it involves converting data from a cross-tabular or multiple response structure into a more manageable columnar format [43].

Pivoting data becomes particularly useful when dealing with crosstab or multiple-response questions where the same data appears in multiple fields. By pivoting the data, we can consolidate and organize it into distinct columns, making it easier to analyze and work with. This transformation simplifies data handling and enables a more efficient representation of the underlying information (Check Figure 5.1).



	A	B	C	D	E	F
1	Respondent	Question 1	Question 2	Question 3	Question 4	Question 5
2	Ken Flerlage	1	4	4	5	2
3	John Smith	5	4	3	3	5
4	Willy Wonka	3	2	5	1	2
5	Fyodor Dostoevsky	5	2	2	1	
6	George Orwell	1	5	5	3	

	A	B	C
1	Respondent	Question	Answer
2	Ken Flerlage	Question 1	1
3	Ken Flerlage	Question 2	4
4	Ken Flerlage	Question 3	4
5	Ken Flerlage	Question 4	5
6	Ken Flerlage	Question 5	2
7	John Smith	Question 1	5
8	John Smith	Question 2	4
9	John Smith	Question 3	3
10	John Smith	Question 4	3
11	John Smith	Question 5	5
12	Willy Wonka	Question 1	3
13	Willy Wonka	Question 2	2
14	Willy Wonka	Question 3	5
15	Willy Wonka	Question 4	1
16	Willy Wonka	Question 5	2

Figure 5.1: What's data pivoting? [39]

When working with data sources such as PDF, Excel, or TXT files in Tableau, the process of pivoting the data is relatively straightforward. You can simply add the data source to Tableau, select the desired columns, and use the pivot function to transform the data from a row-based to a column-based format. This allows you to easily analyze and visualize the data.

However, when dealing with SQL data sources, the pivot functionality may not be directly available. In such cases, you can utilize custom SQL queries to achieve the desired data transformation. Custom SQL queries allow you to write specific instructions to manipulate and reshape the data retrieved from the SQL database. By crafting custom SQL statements, you can perform the necessary pivoting operations or any other required data transformations to prepare the data for analysis in Tableau. By leveraging custom SQL, we had more flexibility in shaping and preparing the data according to our specific requirements.

To effectively visualize and create graphs for certain KPIs like integration and unit test results in Tableau, it was necessary to perform data pivoting using custom SQL queries. To do so we used a specific SQL query:

```
SELECT `unit_test_results`.`id`,
`unit_test_results`.`date`,
`unit_test_results`.`percentage_pass`,
'Run' AS `unittest_name`,
`unit_test_results`.`run` AS `unittest_value`
FROM `bp_unit_test_results`

UNION ALL

SELECT `unit_test_results`.`id`,
`unit_test_results`.`date`,
`unit_test_results`.`percentage_pass`,
'Pass' AS `unittest_name`,
`unit_test_results`.`Pass` AS `unittest_value`
FROM `unit_test_results`
```

```
UNION ALL

SELECT `unit_test_results`.`id`,
`unit_test_results`.`date`,
`unit_test_results`.`percentage_pass`,
'Total Testcase' AS `Unittest_name`,
`unit_test_results`.`Total_testcase` AS `Unittest_value`
FROM `unit_test_results`
```

This query transforms a table called unit_test_results having the attributes (id, date, run, pass, total_test_case, percentage_pass) into a pivoted table having the attributes (id, date, unittest_name, unittest_value).

Furthermore, in certain graphs, it was necessary to derive additional measurements or calculations to achieve the desired graph format. This was accomplished by creating new calculated fields within Tableau.

By defining custom calculations using formulas and functions, we were able to manipulate and transform the existing data to derive new insights and metrics. These calculated fields allowed us to perform complex calculations, aggregations, comparisons, or data transformations that were not directly available in the original dataset. By leveraging Tableau's calculation capabilities, we could incorporate additional dimensions and metrics that enhanced the depth and richness of the visual representations, enabling more comprehensive analysis and interpretation of the data. One of the added calculations is mentioned above:

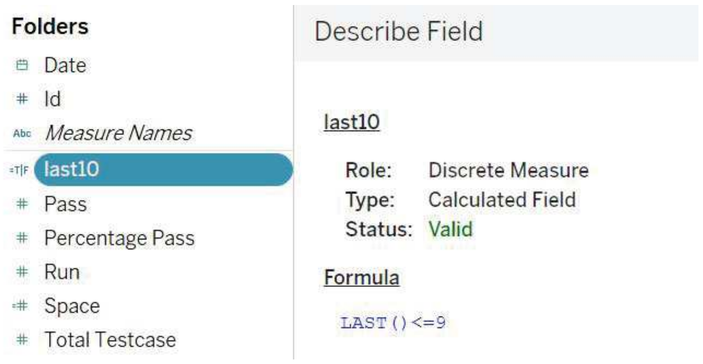
```
Role: Continuous Measure
Type: Calculated Field
Status: Valid

Formula

CASE ATTR([unit_test_name])
    WHEN 'Total testcase' THEN INDEX()+[Space]
    WHEN 'Run' THEN INDEX() +[Space]
    WHEN 'Pass' THEN INDEX() +[Space]+[Space]
ELSE INDEX()
END
```

In certain graphs as well, it was necessary to focus on the most recent data by filtering the last 10 weeks' worth of information. To accomplish this, we implemented a filtering mechanism within Tableau. Specifically, we created a filter that selects the last 10 elements in a table based on a specific sorting criterion, such as a date or a unique identifier Check Figure 5.2.

By applying this filter, the graph displays only the data relevant to the most recent 10 weeks, allowing for more focused analysis and visualization. This filtering capability in Tableau enables us to dynamically adjust the displayed data, ensuring that the graphs accurately reflect the desired time frame and provide meaningful insights into recent trends and patterns.



Role: Discrete Measure
Type: Calculated Field
Status: Valid

```
LAST()<=9
```

Figure 5.2: Last10 filter in Tableau?

5.1.1.3 Publishing and Sharing the dashboard

The main purpose of publishing a Tableau dashboard is to make it accessible to others, and it involves several steps. First, ensure that the dashboard is complete and functioning properly. Then, configure the permissions to determine who should have access. Next, upload the dashboard file and data sources to Tableau Server or Tableau Online, providing a title, description, and relevant tags. Set refresh schedules because the data needs to be regularly updated. Thoroughly test the dashboard before sharing it with others, ensuring all functionalities work correctly. Share the dashboard with the intended audience, providing them with access links or embed codes. Encourage collaboration and feedback by enabling features such as commenting and subscriptions. Monitor the usage and performance of the published dashboard, addressing any issues or errors promptly. Regularly update the dashboard as needed to accommodate changing requirements.

5.2 Dashboard automation using Jenkins continuous integration

5.2.1 Version Control with Git and Bitbucket

5.2.1.1 What is Git and Bitbucket?

Git is a widely used distributed version control system (VCS) that facilitates collaboration and tracks file changes among multiple contributors. It enables efficient teamwork by managing different versions of source code, merging updates from team members, and maintaining a comprehensive history of project development. With features like branching, merging, and tagging, Git empowers developers to handle code effectively and work on different aspects of a project simultaneously.

Bitbucket, on the other hand, is a web-based service designed for hosting Git repositories. It provides a platform for teams to store, manage, and collaborate on their Git repositories. With capabilities such as access control, pull requests, code reviews, and issue tracking, Bitbucket enhances team collaboration and streamlines the software development process. It offers the flexibility of both self-hosted and cloud-based solutions, enabling teams to choose the deployment option that best fits their needs.

5.2.1.2 ASPICE dashboard repository setup

To prepare the setup for the ASPICE dashboard, we considered these steps for the configuration of the repository:

- Repository Creation: Creation of a new repository in bitbucket dedicated to the ASPICE dashboard scripts. We proceeded with setting up the local Git repository and establishing the connection between the local repository and the remote Bitbucket repository.

- Branching strategy: The ”main” branch, formerly known as the ”master” branch, is typically reserved for stable and production-ready versions of the codebase. It represents the latest stable release that is deployed to production environments. To facilitate ongoing development and collaboration, a common practice is to use a separate branch called ”develop” for active development work. This branch serves as the primary integration point for ongoing updates and new features. Developers work on this branch to implement and test changes, ensuring they do not directly modify the main branch until the changes have been reviewed and validated.

- Folder Structure: The ASPICE dashboard implementation involves utilizing various tools such as CI, JIRA, JAMA, and Rectify. For each of these tools, dedicated Python scripts are developed to fetch data and integrate it into the dashboard. In the Git repository, separate directories are created to organize these scripts, with each directory containing the respective code and configuration files. This structure ensures a clear separation of the scripts and allows for efficient management and maintenance of the codebase. Developers can easily navigate to the relevant directory to access, modify, or update the specific scripts related to each tool.

- Documentation and Guidelines: To facilitate the usage and maintenance of the ASPICE dashboard scripts, thorough documentation has been prepared in the form of a README file. This README file provides instructions on how to run the scripts and configure the necessary configuration files. Additionally, the code itself is extensively commented on, ensuring clarity and comprehensibility for developers. The comments provide explanations and context for different sections of the code, making it easier for others to understand and make any necessary modifications or enhancements. This combined approach of a detailed README file and well-commented code promotes transparency, collaboration, and ease of use for the entire development team.

5.2.2 Continuous Integration and Deployment (CI/CD) of ASPICE dashboard

5.2.2.1 What is (CI/CD)?

Continuous Integration (CI) is a software development practice that emphasizes frequent integration of team members' work. It involves integrating changes at least once a day, leading to multiple integrations throughout the day. These integrations are automatically verified through builds and tests, aiming to quickly detect any integration errors. By adopting Continuous Integration, teams can experience reduced integration problems and achieve faster and more cohesive software development [44].

Previously, team developers used to work individually for extended periods, merging their changes to the master branch only after completing their work. This approach led to challenges and delays in merging code changes, resulting in a buildup of bugs that remained unaddressed for a significant period. Consequently, it became more challenging to deliver timely updates to customers.

The abbreviation "CD" in CI/CD stands for continuous delivery and/or continuous deployment, two closely related notions that are commonly used simultaneously. They require the automation of additional steps in the software development pipeline, but they may also be used separately to demonstrate the level of automation used [45].

Continuous delivery (CD) often entails automatic testing and publishing modifications made by a developer to a repository, such as Bitbucket. The updates can then be deployed to a real production environment by the operations team. This strategy tries to solve difficulties of visibility and communication between development and business teams. The basic purpose of continuous delivery is to render new code deployment as simple as feasible.

Continuous deployment corresponds to the automated release of the changes made by developers from the repository to the production environment. That is where the product is accessible to consumers. This method addresses the issue of manual procedure, which slows down application delivery. The continuous deployment automates the stages after continuous delivery in the pipeline Check Figure 5.3.

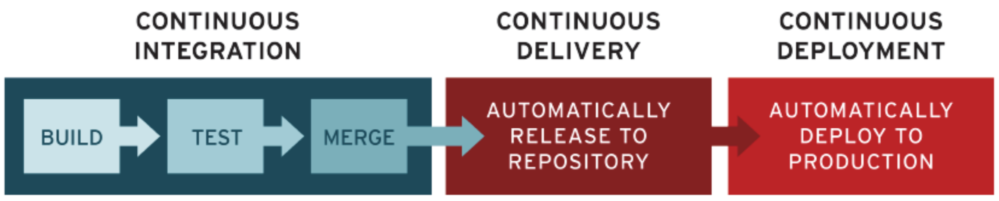


Figure 5.3: CI/CD

5.2.2.2 CI/CD tools

CI/CD solutions can assist a team in automating their development, deployment, and testing processes. Some tools specialize in integration (CI), others in development and deployment (CD), and still others in continuous testing or related services.

The Jenkins automation server is one of the most well-known open-source CI/CD technologies. Jenkins is built to handle everything from a simple CI server to a full CD hub. It offers a framework for automating numerous operations such as software development, testing, and deployment. Jenkins enables developers to seamlessly merge code changes into a common repository and trigger tests and builds to assure the quality of the code. It integrates with several version control systems, testing frameworks, and deployment tools, allowing for greater adaptability and customization of CI/CD pipelines. Jenkins is well-known for its robust plugin environment, which allows users to enhance its capabilities and link it with a wide range of services and tools.

5.2.2.3 ASPICE dashboard integration

The primary objective of this section is to integrate the data fetching scripts into a Jenkins pipeline to ensure their execution every week, following the completion of all tests. This is crucial because the dashboard data is updated once a week. By incorporating the scripts into the Jenkins pipeline, we can automate the process and ensure that the latest data is fetched and incorporated into the dashboard on a regular and scheduled basis. This integration helps streamline the workflow, improve efficiency, and maintain the accuracy and timeliness of the dashboard information. The steps for the integration are:

- Set up the Jenkins pipeline: We already have existing Jenkins pipelines set up for each project (Check Figure 5.4). To integrate the dashboard functionality, we need to incorporate an additional stage in the Jenkinsfile for each project's pipeline. This new stage will be responsible for executing the data-fetching scripts and updating the dashboard after the testing part. Adding this stage to the Jenkinsfile ensures that the dashboard update process is seamlessly integrated into the existing project pipelines. This approach allows us to leverage the infrastructure and automation already in place, making it easier to maintain and manage the dashboard updates alongside the project development and testing processes.

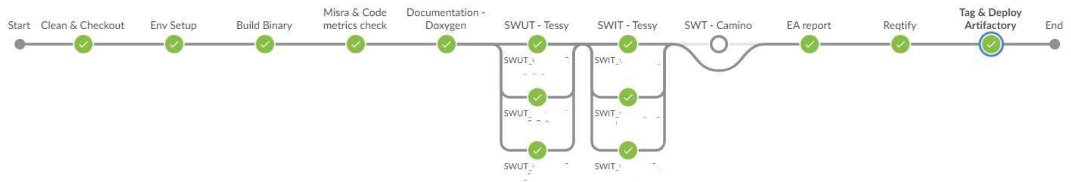


Figure 5.4: Project's Jenkins Pipeline

- Install necessary plugins: We took the necessary steps to ensure the smooth execution of the Python scripts by installing all the required libraries and dependencies and updating the environment. Instead of creating a new environment from scratch, we added the necessary dependencies to the existing environment. This approach allowed us to leverage the existing setup and avoid duplicating efforts. By updating the environment, we made sure that all the required libraries were installed and accessible, enabling smooth execution of the Python scripts within the Jenkins pipeline.

- Create a stage for data fetching: Within the Jenkins pipeline, we defined a stage specifically for fetching data using Python scripts. This stage will be executed after the tests have been completed.
- Define the script execution step: Within the data fetching stage, use a Jenkins step (such as ”sh” for shell execution) to execute the Python script. Provide the path to the Python script and any required arguments.
- Schedule the pipeline: Configure the Jenkins pipeline to run every week, specifying the desired day and time for execution. This can be done using Jenkins' built-in scheduling capabilities or with the help of plugins like the ”Pipeline Utility Steps” plugin.
- Test and monitor the pipeline: Validate the pipeline by running test builds and verifying that the Python scripts are executed correctly. Monitor the pipeline's execution to ensure it runs reliably on a weekly basis.

5.3 ASPICE Dashboard Presentation

The development and deployment of the dashboard have been successfully completed, making it readily available for use. The user interface is designed to be intuitive and user-friendly, ensuring a seamless experience for all users. The graphs and visualizations within the dashboard are carefully crafted to be easily understood and interpreted, providing valuable insights at a glance. In the next Figures, we will present a selection of these graphs, highlighting key metrics and trends to further enhance our understanding of the data.

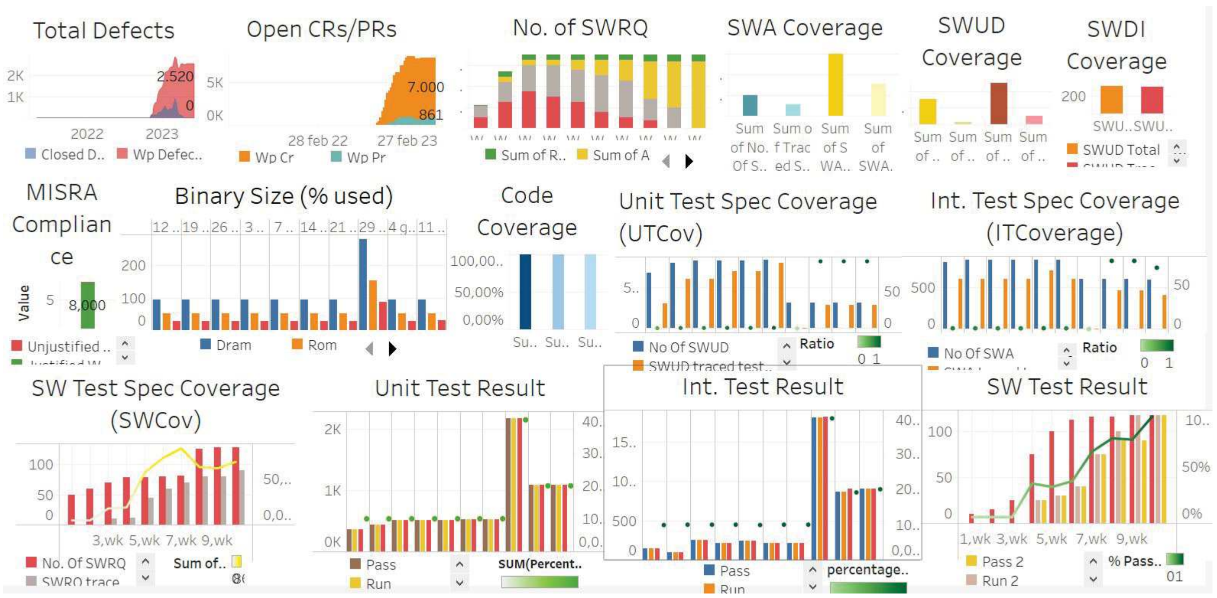


Figure 5.5: Primary KPIs visualization

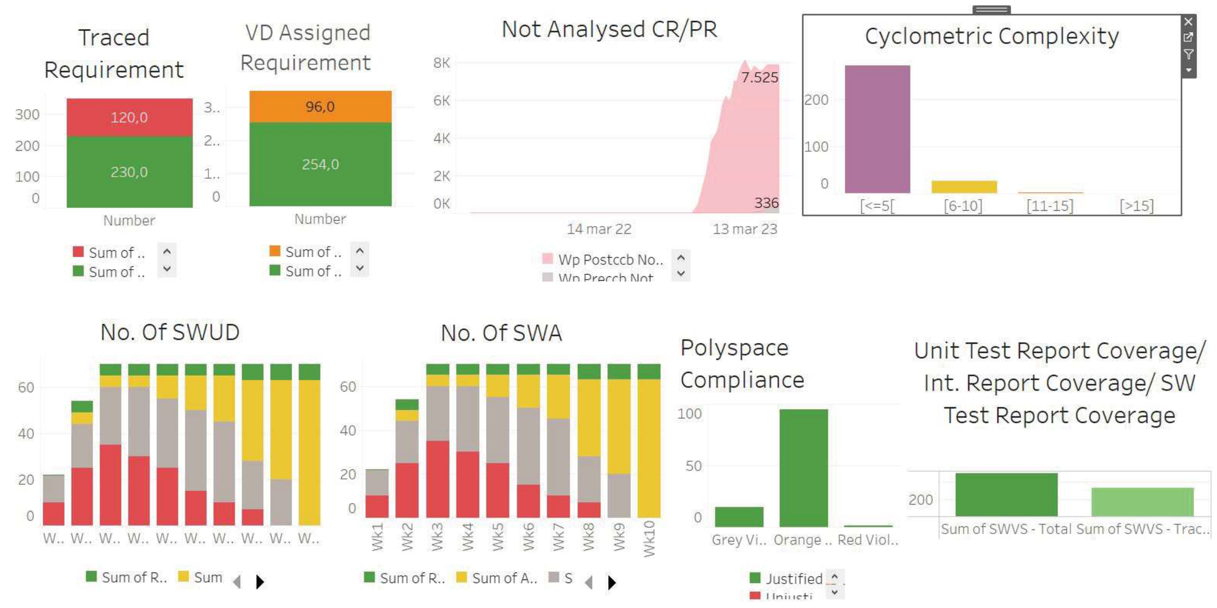


Figure 5.6: Secondary KPIs visualization

Conclusion

Throughout this chapter, we have explored the successful development and deployment of the ASPICE dashboard. This powerful tool offers valuable insights and essential metrics for efficient project monitoring and informed decision-making. We have showcased a curated collection of insightful graphs that demonstrate the dashboard's ability to provide clear and actionable information. With its user-friendly interface, the dashboard ensures effortless data interpretation and empowers users to unlock the full potential of their project data.

General Conclusion

General Conclusion

This report outlines the work completed during a six-month graduation internship project titled ”Automated Environment for the Analysis of ASPICE Compliance for Automotive FW” at Infineon Technologies Padova Italy. The report begins with a project description, presenting the hosting organization, the problem statement, and the main ideas proposed for the solution.

A comprehensive exploration of ASPICE and its associated aspects is then undertaken. It starts with an introduction to ASPICE, emphasizing its significance in software development and quality assurance. The report proceeds to discuss the various processes and activities involved in ASPICE compliance, including requirements engineering, software testing, and configuration management.

Furthermore, the report examines the tools and techniques used to support data fetching, covering topics such as APIs, web scraping, and query analysis. Notably, the database design component of this thesis plays a vital role in effectively managing and organizing ASPICE-related data. Through meticulous entity-relationship modeling, a robust and scalable database schema is developed, providing a solid foundation for data storage, retrieval, and analysis. This thoughtful approach to database design significantly contributes to the overall success and value of the ASPICE implementation.

In addition to these aspects, this thesis encompasses the development and implementation of a robust ASPICE dashboard. This dashboard serves as a centralized platform for monitoring project performance and visualizing key performance indicators. Data fetching scripts are integrated, leveraging powerful visualization tools like Tableau, and harnessing the capabilities of Jenkins for seamless automation.

The findings and insights derived from this research contribute to a broader understanding of ASPICE and its practical implementation in software development organizations. By embracing ASPICE principles, organizations can enhance their development processes, improve product quality, and deliver software solutions that meet or exceed customer expectations.

Overall, this thesis emphasizes the importance of adopting industry standards and best practices, such as ASPICE, to drive excellence in software development. By adhering to established frameworks, organizations can foster a culture of continuous improvement and deliver high-quality software products that align with customer needs and market demands. Throughout this internship, I have had the valuable opportunity to work and familiarize myself with the ASPICE standard, while also gaining insights into the fields of data extraction, database design, and continuous integration. I am grateful for the support of both my supervisors and university mentors, enabling me to apply the knowledge gained during my studies in a flourishing and real-world work environment.

References

[1] Timothy J Sturgeon et al. "Globalisation of the automotive industry: main features and trends." In: *International Journal of Technological Learning, Innovation and Development* 2.1-2 (2009), pp. 7–24.

[2] Helmut K Berg, Prakash Rao, and Bruce D Shriver. "Firmware quality assurance." In: *Proceedings of the June 7-10, 1982, National Computer Conference*. 1982, pp. 3–10.

[3] *The Automotive SPICE: Simple Guide to Get Started*. URL: https://www.tagueri.com/en/the-automotive-spice-simple-guide-to-get-started/ (visited on 01/27/2023).

[4] *Infineon Technologies*. URL: https://www.infineon.com/ (visited on 01/16/2023).

[5] Edward Kit and Susannah Finzi. *Software Testing in the Real World: Improving the Process*. ACM Press/Addison-Wesley Publishing Co., 1995.

[6] PK Ragunath et al. "Evolving a new model (SDLC Model-2010) for software development life cycle (SDLC)." In: *International Journal of Computer Science and Network Security* 10.1 (2010), pp. 112–119.

[7] *Automotive SIG: Automotive SPICE, Process Assessment Model (PAM)*. URL: https://www.automotivespice.com/ (visited on 01/27/2023).

[8] *A Guide to Automotive SPICE*. URL: https://spyro-soft.com/aspice-101-a-guide-to-automotive-spice (visited on 02/03/2023).

[9] *Key Performance Indicators*. URL: https://www.investopedia.com/terms/k/kpi.asp (visited on 02/15/2023).

[10] *Automotive SPICE as part of VDA Scope*. URL: https://www.kuglermaag.com/automotive-spice/ (visited on 02/15/2023).

[11] Bhaskar Vanamali. "Software requirements analysis - SWE.1 in Automotive SPICE." In: *KUGLER MAAG CIE GmbH* (2020).

[12] *The essential guide to requirements management and traceability*. URL: https://www.jamasoftware.com/requirements-management-guide/measuring-requirements/status-requests-changes (visited on 02/25/2023).

[13] Kevin Brennan et al. *A Guide to the Business Analysis Body of Knowledge*. IIBA, 2009.

[14] Pieter Hooimeijer and Westley Weimer. "Modeling bug report quality." In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated Software Engineering*. 2007, pp. 34–43.

[15] Bhaskar Vanamali. "SOFTWARE ARCHITECTURAL DESIGN – SWE.2 in Automotive SPICE." In: *KUGLER MAAG CIE GmbH* (2020).

[16] Bhaskar Vanamali. "SOFTWARE DETAILED DESIGN AND UNIT CONSTRUCTION – SWE.3 in Automotive SPICE." In: *KUGLER MAAG CIE GmbH* (2020).

[17] Christof Ebert et al. "Cyclomatic complexity." In: *IEEE Software* 33.6 (2016), pp. 27–29.

[18] *What is Cyclomatic Complexity?* URL: https://www.tutorialspoint.com/software_testing_dictionary/cyclomatic_complexity.htm (visited on 02/28/2023).

[19] Christian Guß. "How to prove that your C/C++ code is safe and secure." In: *Journal Name* Volume.Number (Year), Pages.

[20] *What Is MISRA? An Overview of MISRA Coding Guidelines and Compliance.* URL: https://www.perforce.com/resources/qac/what-misra-overview-misra-standard (visited on 03/03/2023).

[21] Bhaskar Vanamali. "SOFTWARE UNIT VERIFICATION – SWE.4 in Automotive SPICE." In: *KUGLER MAAG CIE GmbH* (2020).

[22] Marko Ivanković et al. "Code coverage at Google." In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2019, pp. 955–963.

[23] *Difference between statement and branch coverage.* URL: https://programmerbay.com/distinguish-between-statement-coverage-and-branch-coverage/ (visited on 03/25/2023).

[24] Klaus Hoermann. "SOFTWARE INTEGRATION AND INTEGRATION TEST – SWE.5 in Automotive SPICE." In: *KUGLER MAAG CIE GmbH* (2022).

[25] *Unit test Vs Integration test.* URL: https://circleci.com/blog/unit-testing-vs-integration-testing/ (visited on 04/12/2021).

[26] Bo Zhao. "Web scraping." In: *Encyclopedia of big data* (2017), pp. 1–3.

[27] *Parse HTML files.* URL: https://www.kodeco.com/2899-how-to-parse-html-on-ios (visited on 04/05/2023).

[28] Joshua Ofoeda, Richard Boateng, and John Effah. "Application programming interface (API) research: A review of the past to inform the future." In: *International Journal of Enterprise Information Systems (IJEIS)* 15.3 (2019), pp. 76–95.

[29] *What is an API?* URL: https://www.postman.com/what-is-an-api/ (visited on 04/11/2023).

[30] Mark Masse. *REST API design rulebook: designing consistent RESTful web service interfaces.* " O'Reilly Media, Inc.", 2011.

[31] *REST Vs SOAP APIs.* URL: https://www.g2.com/articles/what-is-an-api (visited on 04/05/2023).

[32] *What is a database?* URL: https://www.oracle.com/database/what-is-database/ (visited on 03/06/2021).

[33] *Database management systems.* URL: https://www.appdynamics.com/topics/database-management-systems#~1-what-is-dbms (visited on 04/15/2021).

[34] Paolo Atzeni and Valeria De Antonellis. *Relational database theory*. Benjamin-Cummings Publishing Co., Inc., 1993.

[35] Christof Strauch, Ultra-Large Scale Sites, and Walter Kriha. “NoSQL databases.” In: *Lecture Notes, Stuttgart Media University* 20.24 (2011), p. 79.

[36] Elisa Bertino and Lorenzo Martino. “Object-oriented database management systems: concepts and issues.” In: *Computer* 24.4 (1991), pp. 33–47.

[37] Konstantinos Domdouzis, Peter Lake, and Paul Crowther. “Hierarchical Databases.” In: *Concise Guide to Databases: A Practical Introduction*. Springer, 2021, pp. 205–212.

[38] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph databases: new opportunities for connected data*. ” O’Reilly Media, Inc.”, 2015.

[39] *How does a SQL works?* URL: https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-sql/ (visited on 06/01/2021).

[40] John Miles Smith and Diane CP Smith. “Principles of database conceptual design.” In: *Data Base Design Techniques I: Requirements and Logical Structures NYU Symposium, New York, May 1978*. Springer. 1982, pp. 114–146.

[41] Sikha Bagui and Richard Earp. *Database design using entity-relationship diagrams*. Crc Press, 2011.

[42] *Tableau Vs Power BI*. URL: https://www.simplilearn.com/tutorials/power-bi-tutorial/power-bi-vs-tableau#what_is_tableau (visited on 04/25/2021).

[43] *Data pivoting in Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_pivot.htm (visited on 03/05/2021).

[44] Martin Fowler and Matthew Foemmel. *Continuous integration*. 2006.

[45] *Data pivoting in Tableau*. URL: https://www.redhat.com/en/topics/devops/what-is-ci-cd (visited on 05/10/2021).