POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

On the accuracy of estimation effort with SonarQube

Analysis on development of a railway application

Relatori
prof. Marco Torchiano
ric. Antonio Vetrò

Studente
Luca Maina
matricola: S263633

Supervisore aziendale
ALTEN Italia
dott. ing. Mattia Trasforini

Anno accademico 2020-2021

Abstract

Software companies spend a lot of resources in improving software qualities and remove everything that can lead to technical issue. The result of the efforts is the creation of documentation conventions and specific coding standard.

At the same time, these companies need to reduce the time to market for the product in order to decrease the development cost and the global expenses. Such demands entail the developers to resort the use of fast and unrefined solutions that later needs to be refactored. These actions introduce in the development process unwanted cost that are only visible in the future. These activities are referred as "technical debt". The introduction of technical debt implies that the companies have the need to measure the consequences in term of time and expenses in order to avoid unexpected behavior during the development of the product.

A precise measure of the above requested effort is not possible due to the complexity of the projects and the differences between languages and technologies. Several methods have been proposed to retrieve an estimation from a set of objective metrics describing the state of the code. Usually companies adopt static analysis tool to measure software quality and the presence of issues.

Among the different tools available, in this study we choose to adopt SonarQube. The tool analyses the code and check if it is compliant against a set of coding standard and each violation is considered as an item. Each item defines a set of information about itself, and moreover considers the time needed to refactor the code in order to eliminate the violation retrieved. Analysis about the diffusion and the impact of technical debt issues in software system have already been performed. Various studies proposed approaches to obtain a precise estimation of the remediation time. Following the methodologies introduced, this study aims to report the analysis made on a real-word experience developing a proprietary software with closed-source code base. We decide to follow the life-cycle of a proprietary and close-source project because it represent a very common case in the industry.

The goal of the study is to compare the collected data with the results of previous studies. We investigate the accuracy of the estimation on remediation effort proposed by the tool SonarQube related to the actual remediation time invested by the developer team. Consecutively, we evaluate the impact of the tool related to customer indications.

Contents

List of Figures

Chapter 1

Software Quality background

1.1 Technical Debt

"Technical debt" was originally coined by one of authors of the Agile Manifesto, the software developer Ward Cunningham. It is used as a metaphor that refers to different development team's activities to expedite the delivery of a piece of functionality or a project which later needs to be refactored. These choices prioritize the use of fast, easy and limited solution instead of using better solution that takes more time. Cunnigham compare the technical debt as a monetary debt because, if not repaid, it can accumulate "interest" making harder to implement changes for future improvements. As he said: "With borrowed money, you can do something sooner than you might otherwise, but then until you pay back that money, you'll be paying interest. I thought borrowing money was a good idea, I thought that rushing software out the door to get some experience with it was a good idea, but that of course, you would eventually go back and as you learned things about that software you would repay that loan by refactoring the program to reflect your experience as you acquired it." Martin Fowler distinguished four types of technical debt and elaborate a graphical representation based on the behavior of developers and teams reacting to this issue. - Accidental td Appear when the team have no choice to recognize / understand the potential issue because of inside and outcoming force. Mainly there is a lack of competence in concurrence with lack in design and planification. - Deliberate td This kind appear when a team have the capacity to correctly complete the task, but deliberately decide to use a different solution. Most of the time this choice improves the speed in the implementation or reduce the overall cost in the short term but shall be considered during the time. - Prudent td The team consciously take the decision to focus on certain most important aspects of an implementation and not focus on other. - Unintentional td This kind of issue

could emerge during the implementation and represent the research of quality of the team. It could be used to overcome the limit of a chosen design.

1.2 Software quality

In this study we have choose to use SonarQube as static analysis framework because it has many advantages. Firstly, it represents the opportunity to use a well know and extensively used industry standard. Secondly, it is an open-source platform and that is an important element in order to select the most suitable plug-in for the language currently used.

The tool SonarQube is a tool that try to analyses the QUALITY of a software product. Grasp the concept about quality and what quality means, is the key to understand how to build better product, and in this context, software product. [iso 25010] In software engineering software quality refers to:

- `Functional quality:` defines how a product complies the functional specification and could be represented as the degree to describe how correct the product is against his requirements.

- `Structural quality:` defines how a product complies the non-functional specification that support the life cycle of the product.

To define a set of characteristics that are relevant for the greatest majority of the software system, the ISO standardize a model with well defined methodologies in order to evaluate the quality.

The model proposed in ISO documentation is the "Software Engineering - Systema and software Quality Requirements and Evaluation" SQuaRE. These regulations introduce a "Quality in use model" have the scope to support the realization of well-designed specification and evaluation for software-based product and system under different perspective. The model takes in consideration all the life cycle of the product, from the analysis of the requirements to the disposal.

In order to fulfill its purpose, the model define a set of characteristic and for each ones a set of sub-characteristics to a better definition of different domains.

The model does not try to know "what" a software does, but "how" it works.

SonarQube focus the attention under the reliability, maintainability and security aspects of the code because are the better understandable concept that can be abstracted from the language and are not influenced by the necessity of the software's stakeholders.

In this study we are mainly interested in understanding the quality under the maintainability domain more than any other characteristics because of the nature of the project under examination.

1.3 SonarQube framework

SonarQube is a platform developed by SonarSource for the automatic inspection of the code quality. It perform automatic reviews with static analisys of code to calculates several metrics such as lines of code or code complexity and verifies the compliance against a specific set of coding rules defined for different development languages. If the analyzed source code violates a coding rule, SonarQube generates an issue that can be handled by developers. The issues are classified according to three main quality characteristics:

- `reliability` describes the ability of a system or component to function under stated conditions for a specified period of time. The issues under reliability domain are called BUGS. A coding error that will break your code and needs to be fixed immediately.
- `Security` points to the protection against threats. The issues under the domain of security are defined as VULNERABILITIES. A point in your code that's open to attack.
- `Maintainability` is defined as the degree of facility with which system is capable of being retained in, or restored to, serviceable operation. The issues that concerned the maintainability domain are referred as CODE-SMELLS. A maintainability issue that makes code confusing and difficult to maintain.

SonarQube also assigns a severity level to each issue, namely:

- `info`: Neither a bug nor a quality flaw, just a finding that could interferes with code readaility.
- `minor`: Quality flaw which can slightly impact the developer productivity.
- `major`: quality flaw which can highly impact the developer productivity.
- `critical`: a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw. The code must be immediately reviewed.
- `blocker`: bug with a high probability to impact the behavior of the application in production. The code must be immediately fixed.

Given an issue, SonarQube provides an estimation, which is defined as the time to remedy that issue. The assumption is that the technical debt assigned to an item is equal to the remediation time for that item. When estimating the remediation time for a TD item, SonarQube does not specify which are the assumptions, if any, about the developer who should remedy that item —e.g., SonarQube does not specify whether, or not, it assumes that the developer has a certain level of

seniority or familiarity with the source code affected by the TD item. The estimated remediation time includes the time to comprehend the code, if any (e.g., to remove an unused import, there is no need to comprehend the code), and the time to modify the code so as to remove the TD item.

While the severity level, type, and coding rule corresponding to each TD item are explicitly returned by SonarQube, the effort level is not. By bearing in mind the effort levels mentioned in SonarQube's documentation we classified each TD item, based on the estimated remediation time. For most rules, the SQALE remediation cost is constant per issue. The goal of SonarQube is to help define the value of this constant and to unify the way those estimations are done to prevent having some big discrepancies.

Chapter 2

Project Environment

2.1 History of railway signalling

Railway signalling can be defined as all the set of systems that are necessary for direct and manage the railway traffic and is essential to maintain the train security. Since different trains shred the same track, the main purpose of the traffic management is to prevent possible collisions.

This need has been present since the birth of railways in Europe and all the safety system developed during the time shared a same basic concept: Trains cannot collide if they are not permitted to occupy the same section of the track at the same time. For this reason, a track forming a route is divided into a series of block with variable length from 1 up to 10 Km.

During the time, the changing in safety system development has concerned mainly the different method used to signal to the trains the permission to enter in the consecutive block section and the way to control the presence of a train in the blocks

From the beginning of the railway era, in the first middle of nineteen century (1800 - 1850), men were employed to stand on each block and signal the trains drivers by hand or flag. These watchmen have no evidence in order to ensure that the train have cleared the section ahead because of absence of communications between the watchman of the subsequential block. For this reason, if a preceding train have stopped for any reason, the crew of the following train have no clue about the absence of other trains in the same block unless it was clearly visible. For this reason, incidents were common, and several systems were developed to guarantee major safety.

The introduction of a telegraph line permitted the communication between the watchmen, and the introduction of the semaphore starting from the 1830's helped the signalling to the train crew.

During the second half of the nineteen centuries (1850 - 1900) were developed multiple system for detect the presence of a train without the need of a human

operator, such as train circuit and counter axis. These devices were necessary to focus the control on the pseudo-automatically signalling.

In the twenty centuries, the effort of the railway agency of the main countries was focused on creating automatic and centralized control system characterized the railways of the main countries for all the twenties century with the refinement of previous systems and the creation of safety protocols on tracks and on trains.

As railway networks grew up and the traffic increase, national agencies needs to maximize the capacity of the routes increase the number of trains entities able to operate on same route. The need to safely manage a major number of entities raised the necessity of know continuously lot of information of the state of trains. The data need to be collected from the entire railway network, managed in routing center and be transmitted back to each train.

For this necessity complete networks of sensors and actuator were build trackside and on-board, all connected via different physical layers to command and control centers that have the ability to planning, directing, coordinating, and controlling the entities on the field.

Also, the on-board equipment changed to be able to communicate the information received to the crew. For this purpose, different cab signalling born in order to manage directly the speed of each train on a wheel known portion of the route.

During the time, for many different reasons, each national agency developed a set of devices, systems and protocols to the management of the nation railway networks. Such granularity in regulations lead to difficulties in the interoperation of different system, particularly during the passing of national borders.

2.2 The new ERTMS/ETCS standard

The ERTMS/ETCS (European Rail Traffic Management System / European Train Control System) is the system of standards for management, control and interoperation of signaling for railways. It is conducted by the European Union Agency for Railways and the main purpose is to create a set of standards to enable the interoperability of the rail transport systems in the European Union and promote the exchanges of people and freight.

Since the EU decision in 1996, ERTMS would become the only standard adopted for all the HS/HC lines that are under development in these years.

HS/HC defined as High Speed / High Capacity indicates a type of rail transport that operate significantly faster than the traditional rail traffic. The concept of HS/HC lines indicates:

- high speed lines: tracks that are featured with equipment that permit the safe use of trains designed for a very high-top speed

- high capacity: means the capacity of the system to transport both passengers

and good on the same high-speed lines, and the capacity to safety manage a big number of trains on the same tracks at the same time.

The new framework of ERTMS/ETCS is composed of these main components:

- `EUROCAB` an open architecture for the on-board equipment that elaborate the information received from different source and elaborate and update the breaking curve;
- `EUROBALISE` a discontinuous system for data transmission track side composed by balises fixed between the rails that send fixed or variable telegram of information to the passing train;
- `EURORADIO` a continuous system for data transmission based on the GSM-R for circuit switched network that is used as base of the internet stack for enabling the communication of train and track-side subsystems to communicate using internet protocols.

The trackside equipment can interact with the on-board equipment via different medium based on the ERTMS level choose in the line.

2.2.1 ERTMS Level 1

The L1 is characterized by lateral signal and a discontinuous system of interaction between the train and the ground using the EuroBalises positioned on the tracks as point of information to the train that can read the messages passing over the balises. The balises can store information about their position, the status of the route and other "static" information as some "variable" information, repeating also semaphore's signals, that the train subsystem can use for reconstruction the breaking curve. This level allows the interoperation of the ERTMS with other national protocols.

2.2.2 ERTMS Level 2

The L2 has the peculiarity to work without lateral signals, and it is currently adopted in the Italian "Alta Velocità" trails. The communication between the train and the ground is established by radio transmission using the EuroRadio protocol. The train sub-system shall collected the static information about the position and the status of the track from the balises and the transmit the data elaborated from the EVC to the CandCS that can have a continuous control of the status of the train and can transmit back the signal and the authorization specifically to the single train. At this level the infrastructure positioned trackside resulted very limited and the safety logic is mainly inside the train.

2.2.3 ERTMS Level 3

The L3 is under feasibility study because of the safety implication. The idea under experimentation is the capacity of using only the continuously transmission of data with EuroRadio without the need of many track-side equipment. This changing could enable the possibility of use logic variable length blocks instead of today's fixed block leading to a control based on the current utilization of the infrastructure.

2.2.4 ERTMS Level 0

The L0 is defined for interoperability between tracks with the presence of ERTMS equipment and with tracks with regional rules. Under this level 0 tracks, the train sub-system only sends the data generated by the train while the train driver has the responsibility to follow the regional signals.

2.3 On-board equipment

The trainborne equipment is composed of a set of components with different interfaces:

- `EVC (European Vital Computer)`: is the main on-board computer, which safely processes the functions based on data introduced by the driver, by the sensor on-board and by the information received from the trackside equipment.
- `DMI (Driver Machine Interface)`: is the main interface between the driver and the EVC. It is used for display signals and indications in every driver cab and it acquire data.
- `Odometry`: is the component that estimate the position and the speed of the vehicle based on the measure of the covered distance offered from different sensors.
- `HSSS (High Speed Sigeginenalling System)`: the client side of the telecommunication system between the train and the ground center.

2.4 Track-side equipment

The trackside equipment is composed of a set of components that is dislocated on all the track network:

- `ETCS (European Train Control System)`: is the system for controlling the safety and the distance of the train in the routes.

- `RBSs (Radio Block Center)`: they manage to exchange data in a secure way using the telecommunication.

- `KMS (Key Management System)`: is the component in charge of the management of the cryptographic keys to enable secure data communications over Euroradio.

2.4.1 KMS architecture

ERTMS exchange data between RBCs and train and vice versa. To enable the safety of the communication, the messages securely authenticated. In the EuroRadio standard is explained how the encryption works on the messages and describe to use a couple of keys that must be shared between the peers. The KMS system communicate the various keys to all the actors that use EuroRadio, firstly the vehicles and the RBCs. KMS is basically composed of a KMC center, a KMAC entity and a PKI. The KMC (Key Management Centre) as the role of a server that is in charge to generate, store, update and dispatching the authentication keys into the ERTMS domain. The KMC can exchange keys with another peer KMC and with the final KMAC entity as the KMS client on EVC. The PKI is used to the management of the certificate used for authentication of the clients and servers.

2.4.2 Project scopes

The project revolves around the development of a KMS library with support of the client functionalities. The software shall be subsequently integrated in the EVC on-board system, inside the module that manage the EuroRadio communication with the trackside RBCs. The software module shall be compliant firstly with the specifications descripted by the UNISIG consortium. The main requirements derived from the “Subset 137: On-line Key Management FFFIS” document with other specification derived from the customer of the software that interest the external interfaces, the dependencies with the rest of the system and the board specifications.

We agreed with the costumer to develop a software in compliance with ISO standard defined in 50128:2011 "Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems" that specifies procedures and technical requirements for the development of all programmable electronic systems which are used in railway control and protection applications. In accordance with the document our goal is to develop and validate of a software classified with a minimum degree of “Safety Integrity Level”, defined as SIL 0.

At the end of the development we have to deliver to the commissioner the final application composed of the source code generated and the documentation that accompanies the software, that imply the creation of:

- `Software Requirement Specification (SwRS)`: document that describe all the functional and non-functional specification that the software shall comply, as defined in ISO
- `Software Architecture Description (SwAD)`: the document that describe the architecture designed for the software component
- `Software Model Description (SwMD)`: the document with the model that describe the the functionalities and the actors of the product.

A further request from the customer is the need for the source code generated shall follow a set of coding rules used to standardize the format of the code base within a project. These rules established a naming convention for each used variable, the template of the different files differentiating between source and header files. Moreover, the coding standard does not allow the use of some typical language constructs in favor of greater readability or to decrease the possibility of introducing errors. It also defined the tools that could be used for the generation of the codes and the executables. In this case we shall use the C language the standard defines the use of a set of compliance compiler that are able to generate the correct executable format for the board choose.

2.5 Project overview

Following the requirements described in Subset 137 the software implement KMS client functionalities that implies the ability to establish a connection with a defined KMS server, from now on defined as Home Key Management Centre (HomeKMC) and exchange data. To fulfill the functionality requested we decide to divide the software into different module that are focused on specific functionalities. These modules can be categorized as follow.

2.5.1 TCP client

The main network interface shall be able to connect and exchange data using the TCP protocol over the IP protocol. We have to use pre-existing interfaces because the networking resources are managed by external component with the access granted via Application Programming Interface (API) provided by the customer. In this module we plan to manage the connections and the interaction that imply the reading and writing operation on the network via a no-secure channel.

2.5.2 TLS client

In order to achieve confidentiality, authenticity and integrity of the distributed cryptographic material, the TLS protocol has been chosen as a layer over the TCP/IP

stack. The authentication shall be guarantee either by using certificates from the Public Key Infrastructure (PKI) or by using a secret Pre-Shared Key (PSK) during the handshake phase, and the data shall be encrypted during the exchange. While the need of PSK credentials is based on external functionalities, the capacity to manage the certificate requested for the PKI capabilities are completely supported by the component. Another important functionality is the capability of communicate using the OCSP protocol to skip slow and time-consuming operation to check the validity of the certificate chains generate from the infrastructure.

2.5.3 PKI client

This component shall be able to communicate with the CA/RA entity to generate, request and update the private / public keys needed for the certificate public infrastructure. This component shall be able to locally generate the key pair's used for create and sign the certificates, and shall support the capability to communicate with the CA/RA in order to request the generation, the update and the revoke of the trusted certificate used for establish the TLS connections.

2.5.4 Key Manager

The component shall be able to execute the command received from the HKMC and manage the data received. The main operations are storing, removing, installing and updating of the parameters used for the EuroRadio safe communication. The component is also responsible to make the information about the cryptographical material available to the rest of the system via specific data representation.

2.5.5 State machine

This last component is responsible to check the correct execution of the functionalities integrated into a secure system with very strictly time management of the execution of multiple tasks. The use of a state machine ensures also the capacity to control the flow of execution and is very recommended in the ISO standard in order to reduce the possible introduction of dangerous behavior that can impact the total system.

Chapter 3

Case of study

Different studies are made using code base from a variety of open source projects. Despite the extended usage of open source application, another big portion of the market is characterized by the presence of multiple closed source application. Mostly of this closed source projects are focused on the industry and company domain.

The scope of this study is to report the result of a real-life experience to analyse: how accurate can be the remediation time proposed by the tool SonarQube in order to overcome the technical debt issue incurred during the development of a closed source project.

3.1 Company of study

This study has been done in collaboration with Alten Italia and has been based on a newly introduced project.

Alten Italia is a company that offers services in Engineering and Technology Consulting for industrial and services businnes sectors. All the information presented in this thesis has been concorded with the company in order to omit each non-divulgeable information.

The project involves the company to develop a software product for an external costumer, from now on referred as Customer, that have to be used within a system for a railway application.

The Customer work in the field of railway and is specialized in design, develop and supply solutions for the traffic management system. The solutions under development is part of a system designed for the ERTMS/ETCS management system.

3.2 Team composition

On the project was allocated a team composed of three members:

- `Project Architect`: is the person who is responsable for the design the architectural structure of the component and maintains a continuous communication with the client's technical team and the managerial office. He plays the role of team leader taking trace of the task log and facilitating the technical obstacle encountered during developing and has the major experience of the software developing and the tool and methodologies necessary in the process.

- `Software Developer 1`: is the person who as a Verification & Validation background and shares the abilities for the development of the code and the functional test during the implementation. The team member has also experience in managing the documentation requested with the release of the software.

- `Software Developer 2`: is the team component with junior experience in software developing in the requested language and with scholastic background of the public key infrastructure.

The team has been composed for the first time to follow this project and we spent some time in order to know each other abilities and capabilities and make formation activities in preparation to the effective start of the working period.

During this preparation time, the Project Architect introduced the rest of the team in the design phase to analyse the requested requirements and to exchange the ideas behind the architectural de¬cision made and we also worked together in order to create the bases for the documentation.

3.3 Project characteristics

To be able to compare the actual study to the article "On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube" [3] we have to take in consideration the main differences between the projects analised:

- `size of codebase:` the smallest project considered by previous work has a source base of 10 thousand lines of code, while the average is of nearly 30 thousand lines of code. The application developed for this study has a size of nearly 10 thousand lines of code;

- `time of development:` the set of applications of the previous work is composed of already started projects with year of usage in the open-source community. Instead, in our study we started from scratch using effort to write the requirements from the standard documents, the costumer requirements and from the environment for inter-operability in the system;

- `Different coding rules:` in this project the customer explicitly requests to have a source code compliant with a set of code rules in order to correctly

interact with the internal policy and methodologies. We are interested to understand if the coding rules can modify, and reduce, the presence of issue in the development;

- `Coding language:` another difference from other studies is that our codebase is composed in C code, while the rest of the studies uses a lot of java-based project. The different language can probably affect the presence and the diffusion of issue language oriented;

Starting from a brand-new project, we decided to collect data in key moments of the project that corresponds to major release of the software.

3.4 Goal of the study

Relying on the mentioned goals, we formulated the Research Questions RQs:

- `RQ1.a` What is the diffuseness of introduced TD items?
- `RQ1.b` What is the diffuseness of TD items with respect to different severity levels?
- `RQ2` What is the accuracy of TD remediation time?
- `RQ3` What is the impact of TD remediation with respect to customers rule?

With the `RQ1.a` we aim to show the trand of the presence of issue during the life-cycle of the development. We expect the number of issues to increase going from the first to the second iteration, while during the iteration

3.5 Roadmap

We choose to execute the analysis at the end of the implementation phases, close to the delivery date of the interim versions of the software. In detail the timeline has been set with the following milestone:

- `milestone 0 - 01/09/2020`: is the kickoff day of the project when the first line of code has been written. At the same time with the code, we also worked on the creation of the documentation,
- `milestone 1 - 21/11/2020`: in this date we finished the implementation of the functionalities requested for first release, defined as “Alpha_1”, and we stopped to add new features. We focused on clearing the code and remediate to the trial functions created for test purpose,

- `milestone 2 - 12/11/2020`: end of issues correction, pre-release for Alpha_1, second run,
- `milestone 3 - 12/01/2021`: end of coding standard corrections, release Alpha_1,
- `milestone 4 - 06/04/2021`: end of implementation of new functionalities for Alpha_2,
- `milestone 5 - 09/04/2021`: end of issues corrections for, Release for Alpha_2,
- `milestone 6 - 24/04/2021`: end of correction defined by customer and final release.

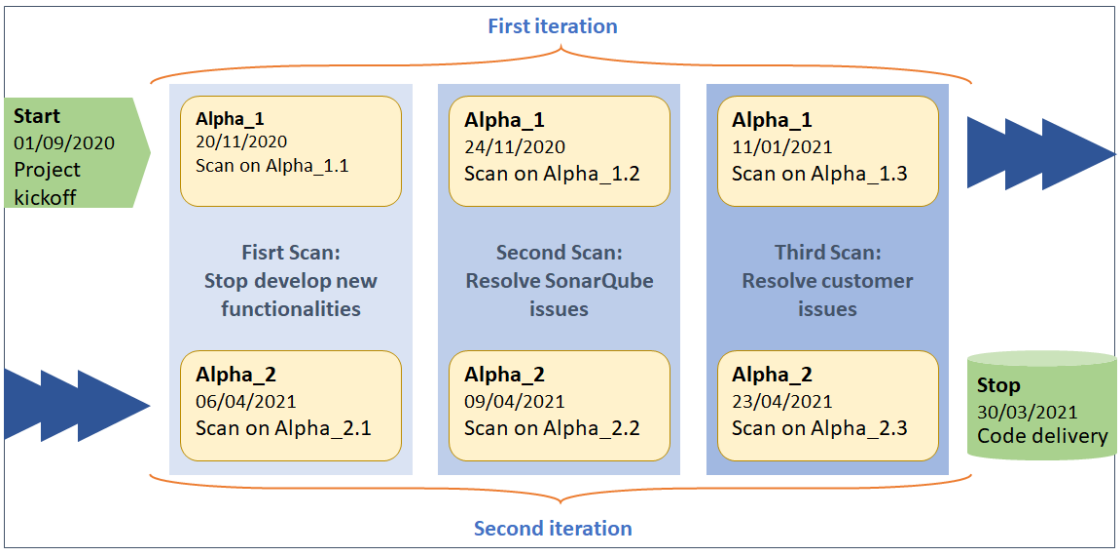

Figure 3.1. Roadmap diagram

During the development life cycle, we encounter the necessity to execute the produced software in a different environment from the final one. To do so we build our component to be semi-independent with all the necessary interfaces between the subsystem stubbed in wrapper functions to be replaced with the customer version. The trial versions were quick and dirty solution needed to test other core functionalities and because of these characteristics we decide to not include the implemented module in the analysis. Also the tools created to test the connections functionalities are not presented, because not part of the release.

Chapter 4

Data Report

4.1 Data collected

Data generated by SonarQube have been collected after each scan using the methods offered by the tool. We query the internal issue database using web API exposed using HTTP request. The resulting data are list of issue with multiple information in JSON format and then refactored in a table for a suitable management. Some information about the architecture of the code have been removed for privacy towards ALTEN Italia.

The obtained table result as:

Rule	Type of Issue	Severity of Issue	Status	Comment Message	Effort Estimeted	Version of Scan
c:S1135	CODE_SMELL	INFO	OPEN	-	0	Alpha_1.1
c:S2095	BUG	BLOCKER	CLOSED	-	5	Alpha_1.2
c:S1079	VULNERABILITY	CRITICAL	OPEN	-	10	Alpha_1.3
c:S1764	BUG	MAJOR	OPEN	-	2	Alpha_2.1
c:S1481	CODE_SMELL	MINOR	OPEN	-	5	Alpha_2.2
c:S1767	BUG	CRITICAL	OPEN	-	5	Alpha_2.3

Figure 4.1. Extract of the final table

The information selected are:

- `Rule`: contains the code of the rule broken to generate the issue. The rules are represented as a string composed as “xxx:yyy” where the “xxx” represent the domain language, in this work we used only C language and so we found only rules for “c: or common-c:”; while the code “yyy” references the current rule:

it could be a name or an alphanumerical code as S123. The numeric value and represent the key of the rules in the internal database. For a detailed explanation of the rule we can visit the site https://rules.sonarsource.com with the complete list of all the rules. To quickly find the single rule we can query directly the URL https://rules.sonarsource.com/c/RSPEC-123 with the current rule numeric code

- `Type of Issue`: classified the issues based on the SonarQube convention. The possible values are:
  - `BUG`: for issue that impact the reliability of the system,
  - `CODE_SMELL`: for issue that impact the maintainability and the readability of the code,
  - `VUNERABILITIES`: for issue that could impact the security of the system.
- `Severity of Issue`: classified the degree of the impact for an issue on the characteristics of the system. The possible values, ordered from the least to greatest, are:
  - `INFO`: neither a bug nor a quality flaw, just a finding that could interferes with code readability,
  - `MINOR`: represent a quality flaw which can slightly impact the developer productivity,
  - `MAJOR`: represent a quality flaw which can highly impact the developer productivity,
  - `CRITICAL`: a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw and must be immediately reviewed,
  - `BLOCKER`: define a bug with a high probability to impact the behavior of the application in production. The code must be immediately fixed.
- `Status`: contains information about the issue lifecycle. Despite the fact that the tool offers more states, we use the tool such as to only have the values:
  - `OPEN`: set by SonarQube on new issues to be resolved,
  - `RESOLVED`: set manually to indicate that the next analysis should Close the issue,
  - `CLOSED`: set automatically by SonarQube for automatically created issues.
- `Comment Message`: contains the comment of the issue created by the scanner with information of the rule broken. It explain the issue in descriptive way, with reference to the actual code.

- `Effort Estimated`: contains the time, in minutes, estimated for remediate the issue. This field is derived from the export to normalize the format of the time.

- `Version of Scan`: contains a string to define the version of the code that have been scanned. The possible values are:

  - `Alpha_1.1`: first version of the code in first analysis iteration,
  - `Alpha_1.2`: second version of the code in first analysis iteration with the correction proposed by SonarQube,
  - `Alpha_1.3`: third version of the code in first analysis iteration with the correction proposed by customer,
  - `Alpha_2.1`: first version of the code in second analysis iteration,
  - `Alpha_2.2`: second version of the code in second analysis iteration with the correction proposed by SonarQube,
  - `Alpha_2.3`: third version of the code in second analysis iteration with the correction proposed by customer,

4.2 Trend in issues diffuseness

The issues that we present are divided in three categories as presented by SonarQube: “bugs”, “vulnerabilities” and “code smells” beacuse the developments team managed each issue in defferent way based on the type.

We expected a defined trend in the two iterations: a greater number of issues at the first scan that will gradually reduce during the successive scans. We foreseen and also an increase in the number of issues between the scans in the first and the second iteration. The described behavior is the most common during a development process because of the increase of funcionalities.

The figures 4.2, 4.3 and 4.4 show the trend of the number of open issues during the phases defined by the roadmap.

For the categories BUG and CODE_SMELL we can notice a suitable representation of the expected behavior, expecially between the scan on Alpha_1.3 and Alpha_2.1 we notice a big increasing in the number of new issues.

Instead, for the category VULNERABILITIES we can see a different trend. Only for this category there are no difference during the code version on the first iteration, but a noticeable decrease in issue occur between the Alpha_2.1 and Alpha_2.2.

The trends are the consequence of the effort expended by the team, following different priorities during the development.

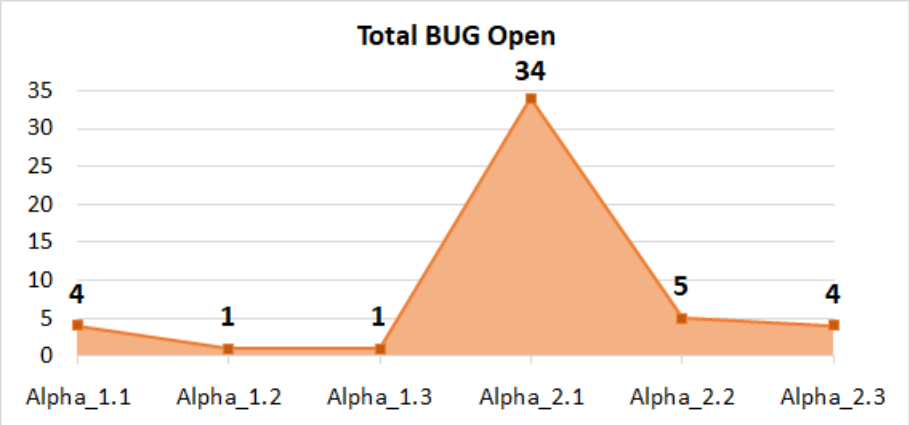

Figure 4.2. Total BUG Open

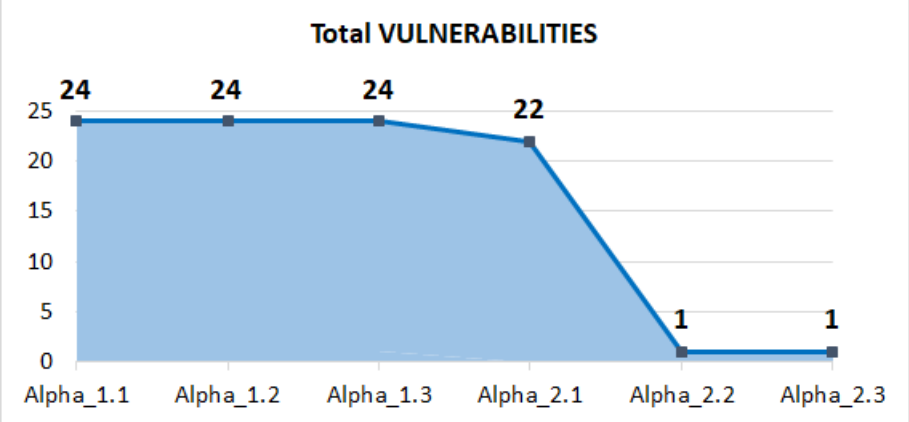

Figure 4.3. Total VULNERABILITIES Open

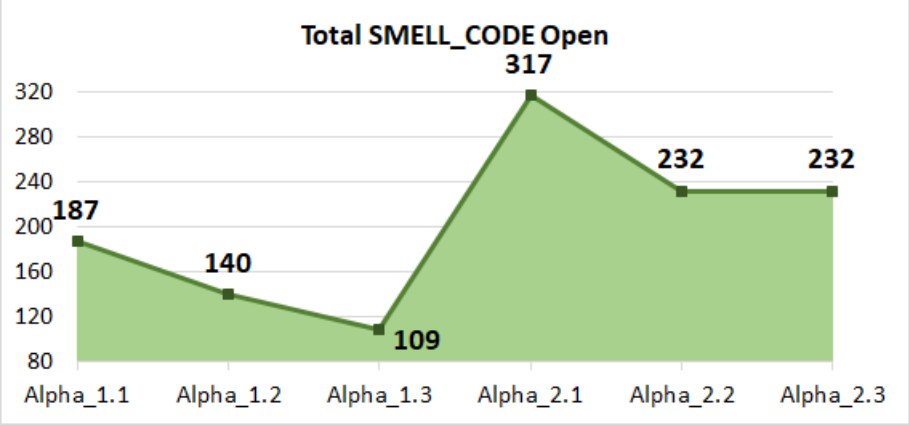

Figure 4.4. Total CODE_SMELL Open

4.2.1 First Iteration priorities

At the end of the first iteration, the main goals were to provide to the costumer an intermediate version of the application with some feature including:

- Implementation of netowrk connection management without security functionalities,
- management of the state machine with the implementation of the main states,
- implementation of the external interface to be called by the rest of the system,
- the parser module to translate the application information over the network,
- a logger interface to generate a readable trace of the application flow.

At this phase of the process we focused on the robustness of the execution and on the readability of the code, in order to improve the comprehension of previous code in future improvements. The issues related to the security of the whole application were not be considered and no issues were affected.

4.2.2 Second Iteration priorities

At the end of the second development iteration, our goals were to provide a working application with:

- Complete and documented external interfaces,
- Complete management of the networking communications with the secure layer,
- Complete management of the all the states concorded to be used in the state machine,
- Complete application logic communication with a server-side simulator,
- Readiness to change some development wrapped functionalities with system specifics ones.

During this phase we focused on the robustness of the application and the documentation of the main functionalities of the code as before, but moreover we spent effort in prepare the solution to be ported on another system with different priorities. By doing so, we have carefully checked for type conversion that were mandatory, all the system dependent functionalities were wrapped to facilitate the exchange at the integration phase.

4.3 SonarQube related issues

4.3.1 Important Bug

Reduction in issues related to “bug” category in order to reduce the unexpected fault during the run of the application. To achieve this result, we focused on the issues that we have mainly encountered at development. Following the specific rule of the C language the issues involves the use of pointers data and types.

The most important issues were:

- `c:S2259` – Access to field is NULL pointer
- `c:S2753` – Implicit conversion of ENUMERATION
- `c:S2095` – Opened file never closed
- `c:S5836` – Undefined pointer value
- `c:S3519` – Out of bound access

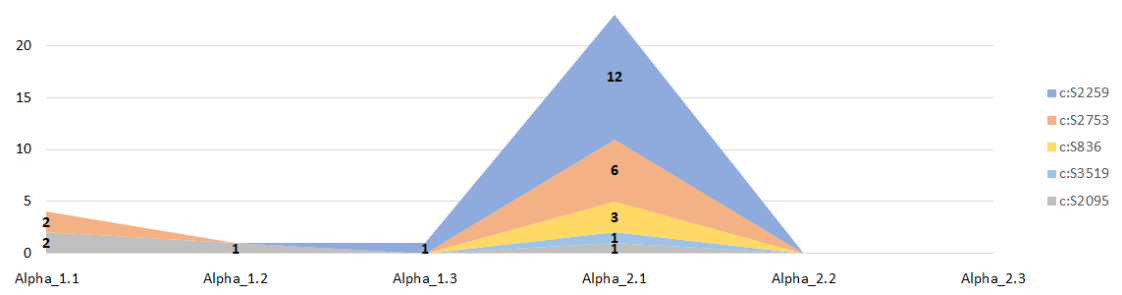

Figure 4.5. Most important BUG

4.3.2 Important Vulnerabilities

Because of the little importance given to security related issue, we briefly analyze the broken rule that generate the warnings. The issues were related to:

- `c:S1079` – Remove use of insecure function printf()
- `c:S1081` - Add a field width specifier to this "%s" placeholder

The rules related to the insecure use of functions in standard libraries didn’t create failure in the execution, and moreover, were located in “external functions” related to the trace functionalities that are not part of the core functionalities requested in the current version, and will be replaced or refactored.

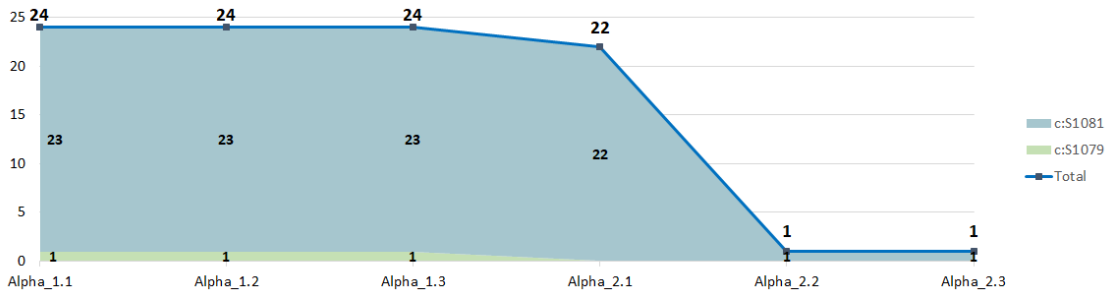

Figure 4.6. Most important VULNERABILITIES

4.3.3 Important Code_Smell

Reduction of the "code_smell" category to improve the readability of the resulting code. The most important issues were:

- `c:CommentedCode` – Comment read as code
- `c:S1854` – Value stored in variable is never read
- `c:S1172` – Unused parameter
- `c:S5276` – Implicit conversion lose precision
- `c:S1135` – Complete the task with TODO

To underline the importance of the readability of the produced code, we choose, with the agreement of the customer, to spend effort in create a documentation directly on the codebase.

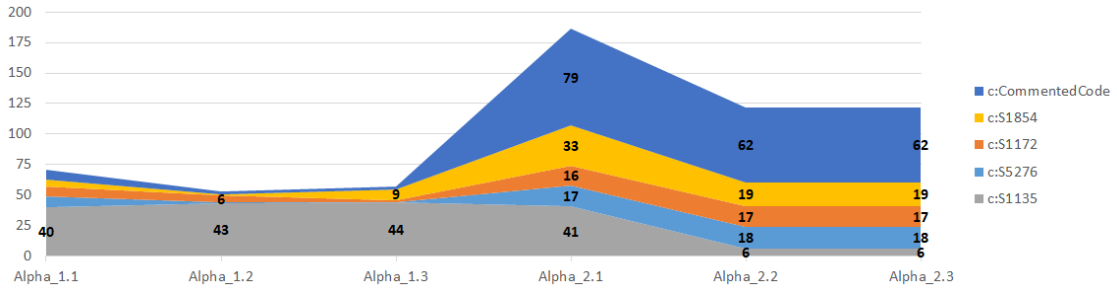

Figure 4.7. Most important CODE_SMELL

4.4 Customer related issues

The customer's report helped use to point out issues related to the coding standard and differences in the external interface to the whole system.

As we can see, these kinds of issues are not well overlapping the SonarQube issues, and the majority of the time expended change very little the rule analyzed by the tool.

The customer's rules are more focused on describe a naming convention and lead the developers to use a methodology called "Defensive Programming", that help to reduce the undesirable behavior of the execution flow. Some of the rules used are:

- At architectural level, a sofwtare is composed on set of well defined component, each component could be composed by a set of software module. Each module must have a file describing the external interfaces called `Module_Ext.h`, a file describing the internal interfaces called `Module_Int.h` and one or more file containing the funtionalities implementation called `Module_Md0.c` up to `Module_Md(N).c`
- Each variables declared in the code must be named pre-pending its type as `BOOL_varName, UINT32_varName, CHAR_varName`
- Each enumeration type must be defined as `typedef enum` and must be named with an ending `_TE`. Each enumeration variables declared in the code must be named as `ENUM_varName`
- If a function generates an error information, the variable used for the error information shall be initialized to the pessimistic result and returned to the caller as:

```
ERROR_TYPE ERROR_resultVar = ERROR_FAIL
ERROR_resultVar = function()
if (ERROR_resultVar == ERROR_OK) {
    // do something
} else {
    // report error
}
return ERROR_resultVar;
```

- The value of a function parameter should be always check if is part of the correct domine,
- Always verify the index before accessing a vector to prevent index out of bound.

Chapter 5

Analysis

5.1 Definition of metrics

In this study we have considered mainly time-related metrics as "global effort estimated" Est_{GE} for evaluate the total estimation and "global actual effort" Act_{GE} to measure the time really needed. These datum are divides by the category related to the issue, while the "global actual effort" includes also the effort for synchronize the team with the code changes. For the team synchronization we considered all the actions related as: meeting for organize the issues and define a common behavior, common code review and all these organizational action. We could define:

$$Est_{GE} = Est_{BUG} + Est_{VULNERABILITIES} + Est_{COSE\_SMELL}$$

Est_{GE} : `Estimated Global Effort`
Est_{BUG} : `Estimated remediation Effort for bugs`
$Est_{VULNERABILITY}$: `Estimated remediation Effort for vulnerabilities`
Est_{CODE_SMELL} : `Estimated remediation Effort for code_smell`

$$Act_{GE} = Act_{BUG} + Act_{VULNERABILITIES} + Act_{COSE\_SMELL} + Act_{SYNC}$$

Act_{GE} : `Actual Global Effort`
Act_{BUG} : `Actual remediation time spent for bugs`
$Act_{VULNERABILITY}$: `Actual remediation time spent for vulnerabilities`
Act_{CODE_SMELL} : `Actual remediation time spent for code_smell`
Act_{SYNC} : `Actual tmie spent for synchronization` $= \sum$ (`team sync actions`)

In order to have data comparable to the study "On the diffuseness of TD" we take in consideration the metric of "Magnitude of Remediation Estimated" MRE define as :

$$MRE = \frac{|Act_{GE} - Est_{GE}|}{Act_{GE}}$$

Also, a mean between the results was calculated and defined as "Mean MRE" MMRE:

$$MMRE = \frac{MRE_{Delta\_1} + MRE_{Delta\_3}}{2}$$

For a further evaluation we also defines the "Impact" Imp of custumer issues remediation with respect to the quality evaluated by SonarQube:

$$Imp = \frac{|Act_{GE} - Est_{GE}|}{Est_{GE}}$$

5.2 SonarQube estimation evaluations

In first iteration, from the data collected during the phase defined as Delta_1, evaluated between the scan of Alpha_1.1 and Alpha_1.2 we found out the following values:

`Estimated Remediation Effort` = `6h 55m`

`Actual Remediation Time` = `3h 11m`

$\texttt{MRE}_{\texttt{Delta\_1}} = 1{,}17$

In second iteration,, from the data collected during the phase defined as Delta_3, evaluated between the scan of Alpha_2.1 and Alpha_2.2 we found out the following values:

`Estimated Remediation Effort` = `21h 46m`

`Actual Remediation Time` = `4h 30m`

$\texttt{MRE}_{\texttt{Delta\_3}} = 3{,}84$

Compare to the previous study [3], our results are very similar to the second half of the case scenario. We can also notice an increasing on the error present in the estimation when the code-base increase in dimension.

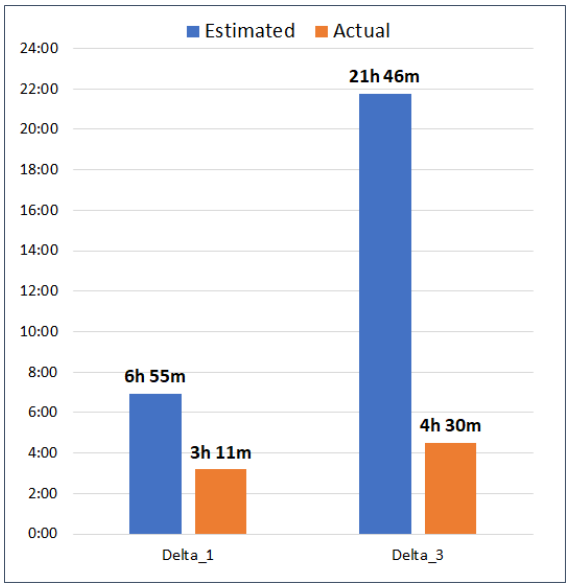

Figure 5.1. Estimated Effort and Actual Time

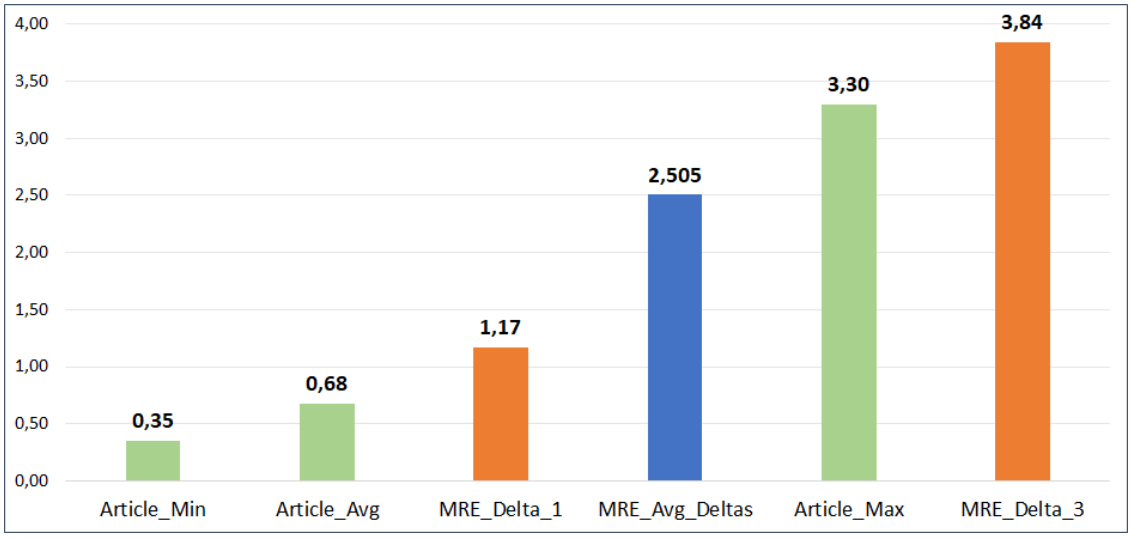

Figure 5.2. Comparison to other result

5.3 Impact of customer evaluation

In the first iteration, for the results obtained during Delta_2 and Delta_4 we would like to observe the impact of the customer issues on the tools estimation measuring the metrics Magnitude Impact MI. In these phases we found that:

$\texttt{Current Effort measured} = \texttt{8h 41m}$

$\texttt{Impact on estimation} = \texttt{41m}$

$\texttt{Imp}_{\texttt{Delta\_2}} = 10{,}95$

In the second iteration, for the results obtained during Delta_2 and Delta_4 we would like to observe the impact of the customer issues on the tools estimation measuring the metrics Magnitude Impact MI. In these phases we found that:

$\texttt{Current Effort measured} = \texttt{13h 30m}$

$\texttt{Impact on estimation} = \texttt{5m}$

$\texttt{Imp}_{\texttt{Delta\_4}} = 161{,}00$

Due to the lack of studies related to the same method of analysis, we are unable to compare current results. We can observe that the impact is really negligible, this behavior can be the result of different causes presented in the next section.

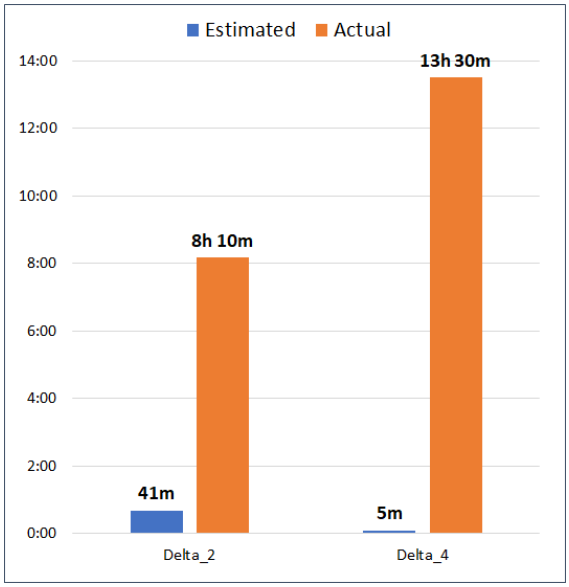

Figure 5.3. Actual Time and Estimated Impact

Chapter 6

Conslusions

6.1 Comment of results

From the metrics analyzed we can assume that the results of this project are similar to the study presented. Some hypothesis on the poor accuracy of the tool should be caused by:

- Presence of multiple instances of the same issue took more time to remediate it in the first time, but then the same strategy could be applied multiple time, according to the surrounding environment, with the reduction of the spent time.
- The presence of parallel coding rules helps us in recreate similar scenarios, make it easier to replicate a solution to the same issue
- Because of the small size of the development team, the person who implements a functionality is the same who refactored the code and have a solid grasp of the surrounding environment.
- In the current project we did not have to generate unit test for each functionality, but a set of component test are run after the changes. The absence of new test surely reduces the remediation time.

On the other hand, the results obtained during the remediation of the customer issues are more difficult to explain objectively. SonarQube offer the possibility to create custom rules to be analyzed during the scan, but could be really time expensive try to find a correct implementation to custom made rules without inserting a great number of false-positive result that overcomplicate the examination of results, leading to introducing an overhead to a tool that try to reduce the development time.

Besides that, the team commonly show the thinks that, basing on previous experiences, spend some time in first place resolving the tools based common issues,

helps in reducing the time spent during the second phase. Reduction that could be as, or even more the time expended in first place.

6.2 Future work

Looking at the changes in our world and in the industry, we can observe that they rely in systems and environments that are growing up in complexity.

To keep up with the progress is necessary to invest a huge amount of effort during the development phase of the projects. Correctly spent this effort is the main goal of each technician, developer, team and department of a technological industry, including railway industry.

In order to manage and organize properly a single person or a group of people during the time needed for developing the future applications, we need the necessary tools and reliable information to take the correct decision at the right time.

When reliable information are not available, we need to produce some estimations that responds to answer similar to: "how much time we need to complete these tasks?" or "how much money it will cost?".

In this work, we considered estimation proposed by the tool SonarQube that is defined as an industrial standard. We compared the obtained results with the state of the art to investigate if the results obtained with open-source software are comparable with proprietary application.

Moreover, we investigated the impact that using such a tool can have in the development of a project with strict custom-defined constraints.

The main outcome of our study is that the current instruments for estimating technical debt are not mature yet. Our results are comparable to the state of the art and underline the importance of using a common coding standard in the team to reduce remediation time of technical debt.

While we analize a single project, further works could applicate this approach of this approach to larger set of data. This approach could be possible done analyzing with more detail the different class of issues proposed by the tool and investigating the result obtainable with the creation of new project-oriented rules.

Besides the fact that we are not able to produce objective result, we are really confident that adopting a tool such SonarQube for the static analysis of the code, really helps the developer to reduce the remediation time, and overall, the total development time.

Acronim list

ERTMS : Euorpena Railway Traffic Management System

ETCS : European Traffic Control Signalling

EUROBALISE : ETCS/ERTMS compliant transponder between rails

EUROCAB : ETCS/ERTMS compliant cabine of the train providing the operating controls

EURORADIO : ETCS/ERTMS compliant radio network equipment with the functions required for a safe communication

IMP : Impact as defined metric

MRE : Magnitude Remediation Estimated

MMRE : Mean Magnitude Remediation Estimated

PKI : Public Key Infrastructure

TLS : Transport Layer Security

TD : Technical Debt

Bibliography

[1] UNISIG, *On-line Key Management FFFIS*, Subset 137 v1.0.0, 2015.

[2] ISO: ISO 50129, *Railway applications - Communication, signalling and processing systems - Safety related electronic systems for signalling*, CENELEC, 2018.

[3] Maria Teresa Baldassarre, Valentina Lenarduzzi, Simone Romano, Nyyti Saarimäki , *On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube*, 2020, doi:10.1016/j.infsof.2020.106377.

[4] *MISRA clarifies safe and secure uses of the C language*, www.misra.org.uk.

[5] SonarSourcce, *SonarQube*, https://www.sonarqube.org.