Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Performance testing for Smart Cards

TITULACIÓ: Doble Grau en Enginyeria d'Aeronavegació i Enginyeria Telemàtica

AUTOR: María Teresa Fernández Mateos

DIRECTOR: Eduard Gracia Villegas

DATA: 8 de Juliol del 2022

Title: Performance Testing for Smart Cards

Author: María Teresa Fernández Mateos

Director: Eduard Garcia Villegas

Date: July 8 th 2022

Overview

One of the most important parts of Research and Development is traceability. The goal of this project is to illustrate this premise by developing performance tests that allow continuous integration and traceability of the product, whilst giving an overview of the processes involved in developing a new product subject to certifications and standards.

From planning, to development and release, this project will focus on the study of the standards and requirements as well as the technical approach and decision making. The project management techniques will also be discussed and explained, to get an insight of the benefits of Agile methodologies in software projects.

The target product of this performance quality testing are Smart Cards and, hence, they will be explained and mastered along this project.

The impact of the performance testing, is having a powerful traceability tool for continuous integration and continuous improvement. This tool plays a fundamental role in the development of a secure product, in an international secure and restrictive environment.

Finally, this project offers a dissertation on Aerospace appliances of Smart Cards and future possibilities for the placement of this product in this field.

INDEX

INTRODUCTION **1**

CHAPTER 1. SMART CARDS **4**

1.1. History and applications 4

1.2. Introduction to Smart Cards 5

1.3. Technology basics 7

1.4. Main Commands 10

1.4.1. APDU Command Parsing 14

CHAPTER 2. PERFORMANCE TESTING **17**

2.1. Requirements and specifications 18

2.2. Technological approach 19

2.3. Test framework 21

2.4. Data visualization 22

2.5. CI/CD 24

CHAPTER 3. PROJECT MANAGEMENT **27**

3.1. Agile 27

3.2. Scrum 28

3.2.1. Roles 29

3.2.2. Ceremonies 30

3.3. Roadmap 31

3.3.1. Pert Chart 31

3.3.1. Gantt Chart 32

3.4. Development 33

CHAPTER 4. AEROSPACE APPLICATIONS **35**

4.1. State-of-the-art of Smart Cards in Aerospace 35

4.2. Future Applications 35

4.3. Smart Card features to enhance UTM 40

CHAPTER 5. CONCLUSIONS **43**

REFERENCES **45**

ANNEX **47**

INTRODUCTION

The development and certification of a new product can be a complex task. If the product is being developed internationally by different teams around the world, this process gains even more complexity, since there must be a unified strategy for integration and quality testing. The goal is to achieve traceability and a seamless integration of the product for certification and for its release to the market.

There are some concepts and methodologies that allow this type of complex technological projects to be developed and become a reality. Concepts such as Continuous Integration and Continuous Development (CI/CD), Agile methodologies, Standards and Specifications will be discussed along this document for further understanding the pillars of an international secure product development.

The process will pay special attention to the project management decisions to illustrate how Agile methodologies and CI/CD have a great impact on the agreeability of the client and the satisfaction of stakeholders all along the project.

Most of these concepts can be implemented to a wide variety of technological products, but this project was devoted to enhancing the development of a new Smart Card product, thus, this technology will be introduced and explained as well in the subsequent chapters. The restrictions and constraints to these niche security products will also be discussed to understand the planning and development phases of this project.

This will be a journey through the standards, technologies and methodologies used in a high security development environment, hence this is a disclaimer that some specific information about the development phase can not be shared in this document for confidentiality reasons.

From planning to release phases, this project will try to cover all the steps and decision making, concluding with a dissertation on the application of Smart Cards for Aerospace, intended to foresee the possible impact of this project in that industry.

A study of the state-of-the-art will be provided about the current applications of Smart Cards in Aerospace and how they have hitherto been linked to ground operations. Nonetheless, some advanced research for drones and traffic management of Unmanned Aircraft Systems (UAS) in urban areas lead to believe that a future market for applications is about to emerge.

Cellular networks seem to be the most fitting technology to support these new traffic management use cases and the industries are preparing for it. Since mobile network operators use Subscriber Identity Module (SIM) cards, to access the network and identify the device and its subscriber, there is strong

evidence to believe that the Smart Card industry shall as well get ready for the future applications in Aerospace.

We will see the impact that Smart Cards, along with cellular network technologies, will have in smart cities and the next generation of air transportation being researched at the moment and supported by big industries, such as package delivery, surveillance, emergency responses and cinematic arts.

Finally, the new possible features and products of Smart Cards, customized to the future scenarios of aviation, will be presented to get a small insight of how new products emerge responding to the needs of an imminent future market.

The aim of this project is, in the first place, to gain knowledge about the Smart Card technology and its implications in modern society; second, to obtain experience in project management, more specifically in the area of Agile methodologies and Scrum framework applied to software development.

We also define a set of more specific goals. The first one consists in developing a set of performance tests, which will be used to enhance the efficiency and quality of the decision making, tracking and development of new Smart Card products. This requires to understand an existing test framework, and the configuration of the tools used for test execution, to design the test cases considering the specifications and the desired coverage, providing an environment for data visualization and integrating the test package in a CD/CI process.

The final object is to link the world of mobile security with the aerospace industry, finding new market opportunities and chances for innovation and creation of new Smart Card products for the future.

The remainder of this document is structured as follows. In Chapter 1, we will introduce the technology of Smart Cards from its origins to nowadays. The history, applications and technology basics are explained. The technology basics will include an overview of the architecture, the communication protocols and the standard commands, concluding with an example of command parsing and decoding.

Chapter 2 focuses on the performance tests developed, justifying the impact of that these tests have in the development of a new secure element product in an international environment. The whole process will be covered from planning and test coverage definition, to the technological approach, the development and the data visualization and integration of the test tool.

The purpose of the Chapter 3 is to show the project management methodologies used and their impact in the agility of the development, the decision taking and the relationship with the stakeholders.

In Chapter 4, applications of Smart Cards in aerospace are discussed and future scenarios are described. This will be related to the research for new possible products and features to be developed.

To conclude, in Chapter 5, the conclusions and future lines of research are discussed.

CHAPTER 1. SMART CARDS

1.1. History and applications

The history of Smart Cards started in 1974 when the French engineer Roland Moreno invented the memory card. The first smart cards were used for telephony, as SIM cards, and for payments, like Automated Teller Machine (ATM) cards. This invention changed the way people interacted, and it helped the advance of telecommunications and globalization. Nonetheless, the first smart card was not quite smart, since they were still passive devices with all the operations performed by the readers (see reference [1]).

Then advances made in microprocessors turned smart cards into active devices, allowing them to perform operations such as encryption of data. Ever since they have been very present in our daily lives, finding more and more applications for them, from payment cards to all sorts of identification cards and accreditations (see reference [2]). As the smart cards became contactless with Radio Frequency Identification (RFID), with a coverage range of a few meters, and Near Field Communication (NFC), with a range of a few centimeters, the ease of use and the applications for them increased.

Smart card technology has evolved to be flexible to adapt to different applications like the embedded SIMs (eSIM), which can be embedded in electronic devices, support several user-profiles and allow easier portability of service for telecom users (see reference [3]).

Administrative documents can also be easily accessed and updated with the help of smart cards, such as identity cards, passports, driving licenses, accreditation, etc. Furthermore, biometric features and digital signatures can be added to increase the security of such documentation.

Payments and finances are still thriving thanks to the use of smart card technology, which allows for a globalized and more available service.

Finally, the increase in the number of devices connected to the Internet of Things (IoT) and Smart Cities is opening new opportunities for smart cards, which will allow secure identification and protection of devices as well as connection to mobile networks for machine-to-machine (M2M) communications.

The current state of Smart Cards and their applications can be observed by the market growth (Figure 1.1), which is distributed in the previously discussed applications. In 2019, the global card production was 3.3 billion units for payment, about 5.2 billion units for telecom providers, 465 million units for mobile phone manufacturers, about 530 million units for government administrative documents, and around 130 million units for industrial and automotive IoT.

Hence, proving that the market of smart cards is expanding, and with the rise of unmanned aerial vehicles, we will soon see many applications in the field of aerospace, as it will be analyzed later on (cf. Chapter 4).

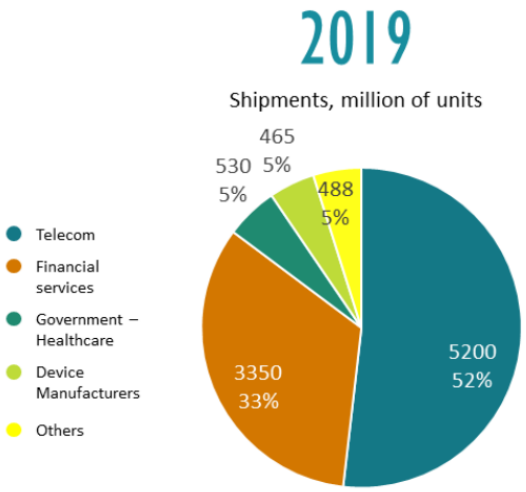


Fig 1.1. Smart Cards market growth in 2019 (see reference [4]).

1.2. Introduction to Smart Cards

Smart cards are devices based on embedded Integrated Circuit Chips (ICC), which can be a microcontroller or only a memory chip. These devices are designed to be tamper-resistant to protect the integrity and confidentiality of the data they hold at the hardware and software levels.

A smart card is always required to be connected with a reader by physical contact or by a remote contactless radio frequency interface, in this way they get powered and activated by the reader.

For the purpose of this assessment, we will focus on microcontroller smart cards, since they can perform processing functions and manage the memory storage which gives support to Java Card technology and hence to the applications that will be discussed later in Chapter 4.

The architecture of a smart card (Figure 1.2) can be divided into three main blocks:

- **I/O System:** establishes the communication with off-card entities. It is controlled by the CPU to ensure that they are standardized, in the form of APDUs (Application Protocol Data Unit).

- **Central Processing Unit (CPU):** executes machine instructions. It does not support multi-threading so, sometimes, another processor is included to perform faster encryption computations.

- **Memory:** there are three types of memory in the card.

- **Random Access Memory (RAM):** volatile memory for temporary storage for fast computation and response.

- **Electrically Erasable Programmable Read-Only Memory (EEPROM):** non-volatile memory, slower than RAM, and used to store applications and their data. It is a limited resource in the amount of reading and writing operations that can be performed on it, so it determines the life cycle of the card in most cases.

- **Read-Only Memory (ROM):** non-volatile memory to store the operating system and other software like encryption algorithms.

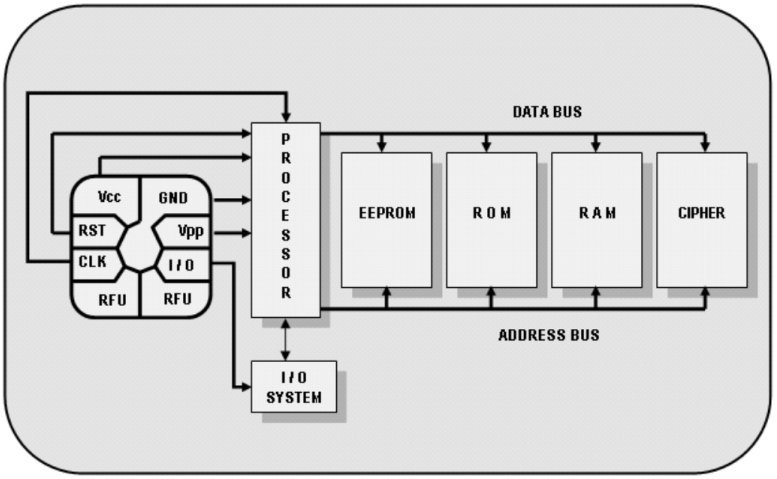


Fig 1.2. Smart Card architecture scheme (see reference [5])

If the smart card supports contactless communications it must also include or be connected to an antenna, and the readability range of these cards depends on the frequency band they use: Low Frequency (LF), High Frequency (HF), or Ultra-High Frequency (UHF).

Some Smart Cards can also be Secure Elements (SE), which applies for any kind of chip that is designed to be protected from unauthorized access and can run applications, as well as store confidential and cryptographic data.

Hence, those Smart Cards which are Secure Elements must meet certain criteria (see reference [6]) regarding security:

- **Hacking and modification attempts must be detected.** The operative system run by the card incorporates means to check the integrity of the data and the operations performed on the card.

- **The Root of Trust (RoT) platform for encryption systems.** The element must be securitized by design in hardware and software.

- **Secure memory for storing private encryption keys and other sensitive data.** Memory access is always controlled and sensitive data is saved encrypted.

- **Secure generation of random numbers for cryptologic operations**. Usually with a specific processor unit to perform crypto operations.

- **Key generation.** Secure key generation to protect data and communication sessions with the off-card entities.

1.3. Technology basics

A Smart Card works in communication sessions established by the card reader, also called the off-card entity, which both powers on the device and activates it.

Then, the communication is carried out by means of small data packets called Application Protocol Data Units (APDU) (Figure 1.3). APDU commands are sent from the off-card entity via a bi-directional channel in half-duplex mode, so data only goes in one direction at a time. Then, after being processed by the smart card, the APDU response is sent.

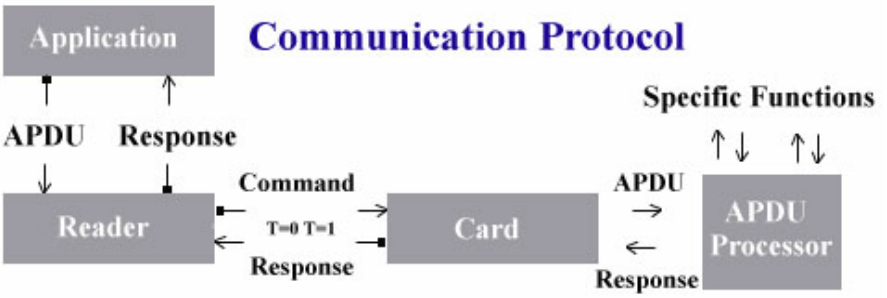


Fig 1.3. Smart Card communication protocol scheme (see reference [7])

There is a standardized set of commands defined by GlobalPlatform specifications (see reference [8]), which are used to manage the applications and the logical secure channels established with the card, as well as the access to resources and operations of the smart card.

Some smart cards support Java Card technology, which allows Java applications, called applets, to run on the card via the Java Card Virtual Machine. This way, multiple applications for ATM secure payment or mobile telecom service providers can be safely stored and executed on the smart card.

Nonetheless, the Java Card API is not designed for multithreading, meaning that commands must be processed individually. Even if multiple logical channels are supported to be opened by the different applications, this would, by no means, make reference to multiple physical channels.

The reason for this is that multiple stacks would be required to perform multithreading, which would cause a high cost in RAM, which is a scarce resource in these applications where everything must be optimized to the maximum to allow for security measures to take place.

Java Card often relies on the GlobalPlatform standard for secure management of applications on the card, such as load, installation, personalization, and deletion of Java Card applets.

These applets must operate in a secure environment that the smart card, as a secure element, must guarantee. The standard GlobalPlatform defines a secure architecture, depicted in Figure 1.4, that ensures hardware and vendor-neutral interfaces to applications and off-card management.

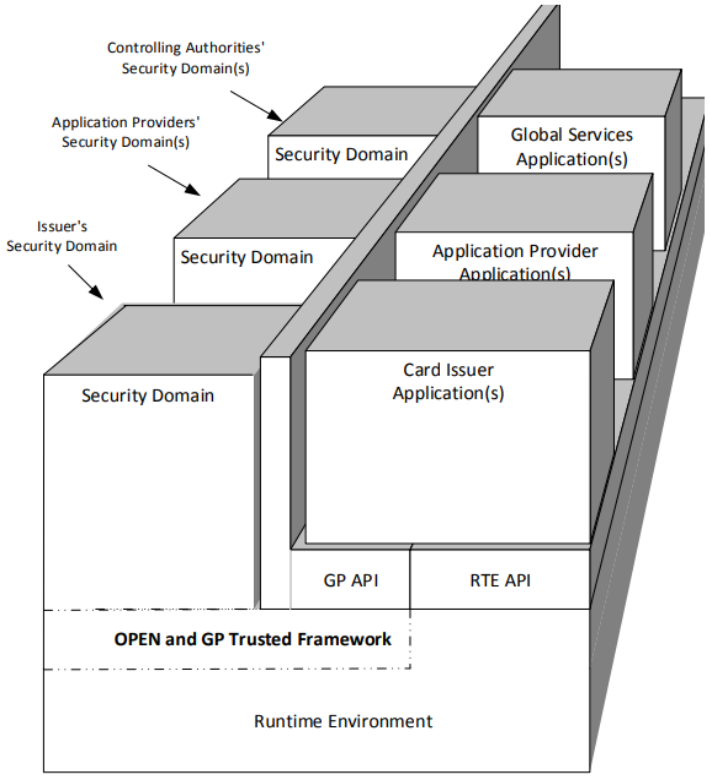


Fig 1.4. GlobalPlatform secure architecture scheme (see reference [8])

In the first layer of the card architecture there is a secure Runtime Environment that includes a hardware-neutral Application Programming Interface (API) to support application portability. It must perform secure memory management by ensuring that no unauthorized information is leaked by any of the secure logical channels.

Trusted Frameworks as GP, provide communication services between Applications. They are usually an extension of the card's run-time environment. It must check the applications' access rules according to their privileges of operation in the card, as well as ensure the proper routing of the commands and the integrity of the responses.

The GlobalPlatform Environment (OPEN) provides an API to applications, command dispatch, application selection, logical channel management, and card content management. It manages the installation of applications, ensuring security principles defined for Card Content management. The OPEN also keeps track of the application that must process the commands received by the

card, also known as command dispatch, considering multiple logical channels that might be open simultaneously. It also manages

The OPEN owns the GlobalPlatform Registry which contains information such as the Executable Load Files, Applications, Security Domain associations, and privileges of each application, in order to manage their life cycle and to regulate all card content management operations, thus ensuring that all changes are authorized by the Card Issuer.

The GlobalPlatform API provides services to applications such as Cardholder Verification Method (CVM), personalization, or security services. It also performs some card content management services. This API will adapt to the OS of the card, which can be MULTOS™ or Java Card™, but for the purpose of this project, the products used for testing run a Java Card OS.

On top of this structure is where the Applications operate. There are different types of applications that can be found in the card.

Security Domains are applications that store the keys and are in charge of the security management of applications, they ensure complete separation of keys between the Card Issuer and multiple other Security Domain providers. Furthermore, they can also perform content management depending on their set permissions, they could install, load, extradite and delete applications or manage secure channels.

They are the on-card representatives of off-card authorities and they support security services such as key handling, encryption, decryption and digital signature generation and verification.

There are several types of Security Domains:

- **Issuer Security Domain (ISD)** is usually the on-card representative of the Card Issuer.
- **Supplementary Security Domains (SSD)** are optional on-card representatives of other Applications.
- **Controlling Authority Security Domains (CASD)** are special kinds of Supplementary Security Domains, on-card representatives of Controlling Authorities.

Security domains tend to form hierarchies in which some SDs are associated with each other and they might support each other providing keys to open secure channels if the SD does not have the right keyset to use.

Global Services Applications provide services to other applications, such as Cardholder Verification Method services, which consist in authentication via methods like PIN or signature.

The central administrator of the card is formed by the GlobalPlatform Environment (OPEN), the Issuer Security Domain (ISD), and the Cardholder Verification Method Services (CVM). This central administrator is known as the Card Manager and it is the maximum authority in the card.

1.4. Main Commands

To operate with the card the main commands must be understood:

- **SELECT:** this command is used to initiate communication with an application of the card. A unique Application Identifier (AID) must be used to select the application. Then a successful status word of two bytes is returned along with some basic information of the application selected, if no errors are found.

  Error code might be returned if the application is not found.

- **AUTHENTICATE:** the authentication must be done in two consecutive steps that conclude with the establishment of a secure channel. The first command is the INITIALIZE UPDATE, where data about the shared keys to authenticate is shared as well as an encryption challenge to authenticate both parties. After a successful INITIALIZE UPDATE, there must be an EXTERNAL AUTHENTICATE to set the security level of the channel from a set of choices in which the command and/or response shall be encrypted and/or signed.

  Error codes might be returned if the authentication failed, or if there are incorrect values in the command data.

- **PUT KEY:** this command is used to add keys or keysets of different cryptographic algorithms to Security Domains that will be used for channel encryption and token or receipt generation. Keys used for channel encryption must form a set of three keys, one for Encryption (ENC) of the whole APDUs, another for Data Encryption (DEK) of sensitive data, and another for Message Authentication Code (MAC). Keys are identified by their Key Version Number (KVN) and their Key Identifier (KID), keys and keysets can be added or replaced with this command.

  Error codes might be returned if a key for replacement is not found, if the cryptographic algorithms are not supported by the card or if there are incorrect values in the command data.

- **GET DATA:** this command is useful to obtain information related to an application or the card. Information such as the AID of the ISD, the list of applications in the card, the current security level or secure channel protocols supported by an application can be obtained with this command. Also, some custom information stored in store data tags can be obtained.

Error codes might be returned if the tags requested are not found or if there are incorrect values in the command data.

- **STORE DATA:** this command is very self-explanatory. It can be used to store data like the AID of the ISD, which would in fact replace its AID for the new one or to store custom information using some of the available range of tags for this purpose. Keys can be stored with this command too, and it is usually applied for keys of long lengths, larger than one ADPU (255 bytes). These commands can be concatenated in store data sessions in which the block number must be sequential and the store data is processed atomically.

  Error codes might be returned if referenced data is not found, if there is not enough memory space or there are incorrect values in the command data.

- **INSTALL:** it is responsible for the installation and hierarchy configuration of the applications in the card. This command has several variants:

  - **INSTALL [for load]:** used prior to the load of a CAP file or executable package to install a new applet.

  - **INSTALL [for install and make selectable]:** to install an application that can be selected.

  - **INSTALL [for extradition]:** to change the hierarchy of application by moving one to another domain or even to its own domain, when extradited to itself.

  - **INSTALL [for registry update]:** to update data associated with an application like its privileges.

  - **INSTALL [for personalization]:** to personalize an application. This is a process that will start with this command and will continue with a set of STORE DATA commands that will set keys and define the capabilities of the application.

  Error codes might be returned if there is a memory failure, if the referenced data is not found, if the application is not found, or if there are incorrect values in the command data.

- **LOAD:** to load a file in the card. This is how applets are installed by first doing an INSTALL [for load] and then loading the CAP file of the applet that will be later on instantiated with an INSTALL [for install].

  Error codes might be returned if there is a memory failure, if there is not enough memory space or if there are incorrect values in the command data.

- **DELETE:** to delete card content such as applications or keys.

  Error codes might be returned if there is a memory failure, if the referenced data is not found, if the application is not found, or if there are incorrect values in the command data.

- **GET STATUS:** to retrieve Issuer Security Domain, Executable Load File, Executable Module, Application or Security Domain Life Cycle status information according to a given match/search criteria. For instance, the life cycle or the privileges can be obtained. This command might have sessions of response chaining in which APDUs are concatenated to return the full response.

  Error codes might be returned if referenced data is not found or there are incorrect values in the command data.

- **SET STATUS:** to modify the card Life Cycle State or the Application Life Cycle State.

  Error codes might be returned if referenced data is not found or there are incorrect values in the command data.

- **MANAGE CHANNEL:** to open and close Supplementary Logical Channels except for the Basic Logical Channel (channel number zero) which can never be closed.

  Error codes might be returned if the secure messaging is not supported, or if the function is not supported.

All these commands are sent via a Secure Channel Protocol operating according to the established Security Level and commands must be sent according to these sessions and their security level. If not, they will receive a status word of error conditions not satisfied.

These levels may require authentication, command encryption and/or MAC and response encryption and /or MAC. The security level is coded in one byte (Table 1.1).

Table 1.1. Current Security Level (see reference [8])

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	-	-	-	-	-	-	AUTHENTICATED
0	1	-	-	-	-	-	-	ANY_AUTHENTICATED
-	-	-	-	-	-	1	-	C_DECRYPTION
-	-	-	-	-	-	-	1	C_MAC
-	-	1	-	-	-	-	-	R_ENCRYPTION
-	-	-	1	-	-	-	-	R_MAC
-	-	-	-	X	X	-	-	RFU
0	0	0	0	0	0	0	0	NO_SECURITY_LEVEL

These commands might also be successful or not, depending on the life-cycle state (Table 1.2) and the privileges that the processing SDs have. Authorized Management (AM) privilege means that the SD will be able to perform content management like installations and deletions.

Nonetheless, Delegated Management (DM) privilege means that the processing SDs would have to request the content management operation to an SD with AM privilege in its hierarchy. In order to do so, a token is generated with a token key previously shared.

Table 1.2. Authorized GlobalPlatform Commands per Card Life Cycle State (see reference [8])

Command	OP_READY			INITIALIZED			SECURED			CARD_ LOCKED	TERMINATED
	AM SD	DM SD	SD	AM SD	DM SD	SD	AM SD	DM SD	SD	SD	SD
DELETE Executable Load File											
DELETE Executable Load File and related Application(s)											
DELETE Application	✓			✓			✓				
DELETE Key											
GET DATA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GET STATUS	✓			✓			✓			✓	
INSTALL [for load]											
INSTALL [for install]											
INSTALL [for load, install and make selectable]											
INSTALL [for install and make selectable]	✓	✓		✓	✓		✓	✓			
INSTALL [for make selectable]											
INSTALL [for extradition]											
INSTALL [for registry update]											
INSTALL [for personalization]											
LOAD											
PUT KEY	✓			✓			✓				
SELECT	✓	✓	✓	✓	✓	✓	✓	✓	✓	See Note 1	
SET STATUS	✓			✓			✓			✓	
STORE DATA	✓			✓			✓				

1.4.1. APDU Command Parsing

APDU commands are parsed and processed following the format and rules defined by the specifications.

The main blocks in the header of each APDU command are:

- **Class byte (CLA):** gives information about the standard used, the logical channel, and the security of the channel.
- **Instruction byte (INS):** gives information about the type of command.
- **Reference control parameters (P1 and P2):** it gives some extra information, it depends on the command.
- **Command Length:** length of the data field in bytes.

Then, we find the data field that contains different information for each command. In the case of a PUT KEY command, it would contain the key or keys.

Here is an example of how the parsing would be done. If we received an ID like the following:

80 D8 00 01 0C 21 80 08 E6 1C C4 F7 1C 01 73 67 00

First, we check the first byte (CLA) and parse it with the specification coding table (Table 1.3).

Table 1.3. CLA byte coding (see reference [8])

b8	b7	b6	b5(*)	b4	b3	b2	b1	Meaning
0	0	0	0	-	-	-	-	Command defined in ISO/IEC 7816
1	0	0	0	-	-	-	-	GlobalPlatform command
-	0	0	0	0	0	-	-	No secure messaging
-	0	0	0	0	1	-	-	Secure messaging – GlobalPlatform proprietary
-	0	0	0	1	0	-	-	Secure messaging – ISO/IEC 7816 standard, command header not processed (no C-MAC)
-	0	0	0	1	1	-	-	Secure messaging – ISO/IEC 7816 standard, command header authenticated (C-MAC)
-	0	0	0	-	-	X	X	Logical channel number

A value 0x80 in the CLA byte would tell us we are processing a GlobalPlatform command in the logical channel 0 and not encrypted.

We would continue parsing the INS byte 0xD8, which means we are processing a PUT KEY command as the coding table in the specifications indicates (Table 1.4).

Table 1.4. PUT KEY command message (see reference [8])

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'D8'	PUT KEY
P1	'xx'	Reference control parameter P1
P2	'xx'	Reference control parameter P2
Lc	'xx'	Length of data field
Data	'xxxx..'	Key data (and C-MAC if present)
Le	'00'	

Then we would decode the P1 byte according to the PUT KEY decoding table (Table 1.5).

Table 1.5. PUT KEY Reference Control Parameter P1 (see reference [8])

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Last (or only) command
1	-	-	-	-	-	-	-	More PUT KEY commands
-	X	X	X	X	X	X	X	Key Version Number

Since the P1 is 0x00 we know it is a single command, so no chaining APDUs are expected for this command. We also know that it is a new key and not a replacement because the key version number for existing keys ranges from 01 to 7F.

Then we would decode the P2 byte according to the PUT KEY decoding table (Table 1.6).

Table 1.6. PUT KEY Reference Control Parameter P2 (see reference [8])

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Single key
1	-	-	-	-	-	-	-	Multiple keys
-	X	X	X	X	X	X	X	Key Identifier

Since the P2 is 0x01 we know it is a single key and the first key identifier is 01. Then, we see in the length byte of 0x0C that the data field expected is 12 bytes. Next byte is the Key Version Number of this new key, which is 21, and then the rest of data of the key.

We parse the data field with the decoding tables (Table 1.7) (Table 1.8).

Table 1.7. Key Data Field – Format 1 (Basic Format) (see reference [8])

Name	Length	Value	Presence
Key type of first or only key component	1	'00' - 'FE' – see section 11.1.8, Key Type Coding	Mandatory
Length of first or only Key Component Block	1-3	'01' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
First or only Key Component Block	1-n	'xxxx...' (see section 11.8.2.3.2)	Mandatory
...	...	...	...
Key type of last key component (if more than one)	1	'00' - 'FE' – see section 11.1.8, Key Type Coding	Conditional
Length of last Key Component Block	1-3	'01' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Conditional
Last Key Component Block	1-n	'xxxx...' (see section 11.8.2.3.2)	Conditional
Length of key check value	1	'00' - '7F'	Mandatory
Key check value	0-n	'xxxx...'	Conditional

Table 1.8. Key Type Coding (see reference [8])

Value	Meaning
'00' - '7F'	Reserved for private use
'80'	DES – mode (ECB/CBC) implicitly known
'81'	Reserved for historical reasons
'82'	Reserved for historical reasons

And we can see that the Key Type byte being 0x80 indicates that it is a DES key. Following, we have the length of the key, which is 0x08, the 8 bytes key and, finally, the length of the Key Check Value, which is 0 in this case.

CHAPTER 2. PERFORMANCE TESTING

Time performance, code footprint and memory management are the key factors to always consider when designing and developing smart cards or any secure element's Operating Systems.

Other performance indicators of the card include hardware testing of endurance, which refers to the number of write operations that the card can perform in EEPROM before being terminated.

Whilst scarce memory resources are a constraint for the size of the code and the management of memory during execution, having to design and implement security countermeasures that protect the secure element is a constraint to performance.

Several types of attacks can be carried out on the hardware of an embedded secure element and these need to be taken into account when designing the Operating System of these devices.

The main attacks targeting (see reference [9]) a secure element, the countermeasures of which can affect time performance can be classified in:

- **Invasive attacks:** like fault injection they use active physical elements on the hardware like lasers to provoke thermal changes to skip instructions of code from the stack or to change some bits from the volatile memory.

  These attacks can cause simple jumps of one instruction, long jumps of several consecutive instructions, cross jumps from different parts of the flow being able to access other methods and cause some data leakage. They can also achieve bit flips in stored variables changing their values.

  Some solutions for these issues would be to add double checks, secure counters for flow control or return strong secure check values in critical methods.

- **Non-invasive attacks:** side-channel attacks that exploit weak channels in which hidden information may leak in the form of physical phenomena.

  This is a type of eavesdropping in which power consumption, electromagnetic emission, photon emission, or timing can be analyzed leaving no tamper evidence to find about the flow of the code. In this way keys can be discovered and also exception reporting instructions can be spotted to then be jumped with a fault injection.

  Some solutions may imply exception delaying as well power consumption simulators to mask some cryptographic processes.

These countermeasures are usually related to critical operations performed on the card, in which sensitive data is accessed, such as content management and cryptographic operations.

Hence, while working on a secure element product being developed by different international teams that work on the different layers of the architecture. It is very important to test the performance of the product on each iteration.

In this way, there is a traceability of the performance cost of every security countermeasure. Then, based on the data collected, the teams can coordinate on the best strategies to apply the countermeasures handling the tradeoff between security and performance.

The idea for this project was to create a test package that would test the time performance of the card both physically and by means of simulation to have a granular and fast mechanism to measure the impact of each security countermeasure applied, as well as future features and performance optimizations developed for the product.

Granularity and modularity is important to reduce the time of execution of the test so there is a wide range of options available for testing. Then, it is possible to test specifically the commands or use cases involved in the new implementation and compare the measure with the records of previous states of the project.

2.1. Requirements and specifications

The requirements for this project were given by the existing framework for product testing developed by Giesecke & Devrient and the restrictions of security needed to maintain the certification of Common Criteria (CC).

The Common Criteria for Information Technology Security Evaluation is an international standard (see reference [10]) for IT security certification that is demanded by national governments, banks and all sorts of security-conscious customers to provide confidence in a highly secure certified product.

One of the restrictions that this certification posed was that the code must not go through any public network and the servers that store the code must be inside highly secure buildings. This makes remote work unavailable so the code must be developed inside the Research & Development department of the company.

The specifications used are provided by GlobalPlatform, an international association that standardizes secure components. The document GlobalPlatform Technology Card Specifications (see reference [8]) was used to understand the behavior of the card and the parsing of the different commands for the tests.

2.2. Technological approach

The technological approach proposed for the development of this project entailed the development of a toolset and software package to extend an existing proprietary test framework that had capabilities to test APDU commands in both card target through a physical reader and a simulation target through logical ports to the open instances.

The simulation target offers an advantage which is having hardware-independent execution times. Due to the fact that the simulation does not add any hardware or physical channel delays, the test execution is much faster and the performance of the OS can be tested independently.

Looking at the difference in the measures taken with different versions, we can see if there has been a performance degradation or improvement in our implementation. This reduces the time of reaction and enhances the frequent use of tests during the development phase, in a test-driven development approach. In this way, we can improve the efficiency and quality of our decision-making and our designs.

Granularity is also a crucial requirement when developing these tests. Test granularity is the level of detail at which your software tests and test cases address your project. We need to have tests that aim at very specific use cases and label them appropriately to have significant metrics of our implementations and changes on the product.

For the creation of these tests, all commands and use cases needed to be listed, according to the specifications, to create a granular test package of independent test cases that would test each of the commands available on the card with different features enabled.

For instance, in the case of the **PUT KEY** command, several scenarios needed to be planned:

- **Put a new key or replace an existing key.** The processing on these two commands and the handling of the memory might be different so they need to be tested and tagged separately.

- **Put a single key or multiple keys.** They need to be tested separately since the code flow is different.

- **Put a key with key check value (KCV) or without KCV.** There is a difference in performance when checking the KCV of a key and cryptography functionalities must be used in that case.

- **Put a key for a secure channel key set or for token or receipt generation.** The keyset for a secure channel involves extra checks of the three keys involved.

- **Put a key for each cryptographic algorithm supported.** The creation of the objects is different to each algorithm; they show different performances.

- **Put keys of different lengths.** The cryptographic operations behave differently in time performance, depending on the length of the key.

- **Put keys encrypted or unencrypted.** Using encrypted keys, like the private key of RSA, takes a different time to process than unencrypted ones.

- **Put a key in different SCPs and security level.** The processing of the APDU is different depending on whether they are encrypted or not, on having MAC, and also on the security level expected for the response.

Different processing and code flow is executed in each case, so different test cases should be created to measure these variants of the command separately.

The decision taken to achieve a better granularity for the tests was to add labels for each command case so that the measurements could be differentiated and, then, obtain metrics for each case, or group them and get average metrics of the command as a whole.

The complete set of test cases cannot be provided in this document, but an analysis such as the one shown with the PUT KEY command was performed for each of the commands defined by the GlobalPlatform specification to create a granular set of tests.

The test package was distributed in four main blocks:

- **Command library:** this was a class with all methods needed to create the APDU commands that will be parsed and sent by the test cases. Here, the specific tags were added and the instruction to print the timestamp under the selected label.

- **Test Scenarios:** these were the classes that grouped test cases with similar precondition requirements, usually ordered by type of command.

- **Test Cases:** These would be the specific tests for each of the cases subject to feature requirements, they would be iterated to obtain several measurements of performance to process into metrics of average, deviation, maximum and minimum. These test cases come from the prior analysis shown previously with the PUT KEY command.

- **Log Parser:** This would be the script executed after the tests to obtain the final summary average, maximum, minimum and deviation metrics by labeled command.

Since this is a test package meant to be executed for several products that are compared against other versions, the test package must also involve some

intelligence when it comes to feature availability. Then, test cases can be executed when it is necessary, depending on the capabilities of the product they are testing.

The test framework used provided a way to achieve this purpose and will be discussed in the next section.

2.3. Test framework

The test framework used is proprietary, and allows for several APDU tests to be executed in both card target or simulation target as mentioned in the previous section.

It provides logs that can be easily interpreted by a proprietary Lean Report Analysis Platform so in case any errors introduced in the products make these tests fail, a report can be uploaded with the errors analyzed and justified with a project implementation task that will conclude re-executing these tests and making sure they pass correctly again.

It has built-in methods to obtain the timestamps of commands received by the readers, which were used in the new test package to obtain the metrics.

In addition, it has a library to map requirements from configuration files. This is how the test is allowed to be feature-aware, since a configuration file is used for every product, stating which capabilities are enabled. Then, some test cases will be skipped if the capabilities tested are not supported by the target.

For instance, some products may not support Elliptic Curve Cryptography (ECC) algorithm, then the tests of PUT KEY or INSTALL with tokens that use this algorithm will be implemented with requirement enabled in the capabilities configuration file.

The fact that the tests can be feature-aware allows these tests to be executed in several products, and provides more traceability apart from agility to the development process and decision making of the projects.

From the beginning, it is possible to have an overview of the optimizations needed, or the distribution of software and hardware responsibilities to enhance performance.

Using data visualization, this can be also used as a continuous development tool to show stakeholders the progress of the product being developed and justify the decisions made.

Hence, as depicted in Figure 2.1, the input of our test framework will be the requirements configuration file, which will configure the execution of our test and decide which test cases must be executed or skipped.

Then, the test framework will use the package and command library developed in this project to execute the test cases in their corresponding test scenarios and will return the test logs as an output.

Then those logs will be interpreted by our log parser, which will, in turn, generate a file that will be compatible with our data visualization tool to obtain the plots.

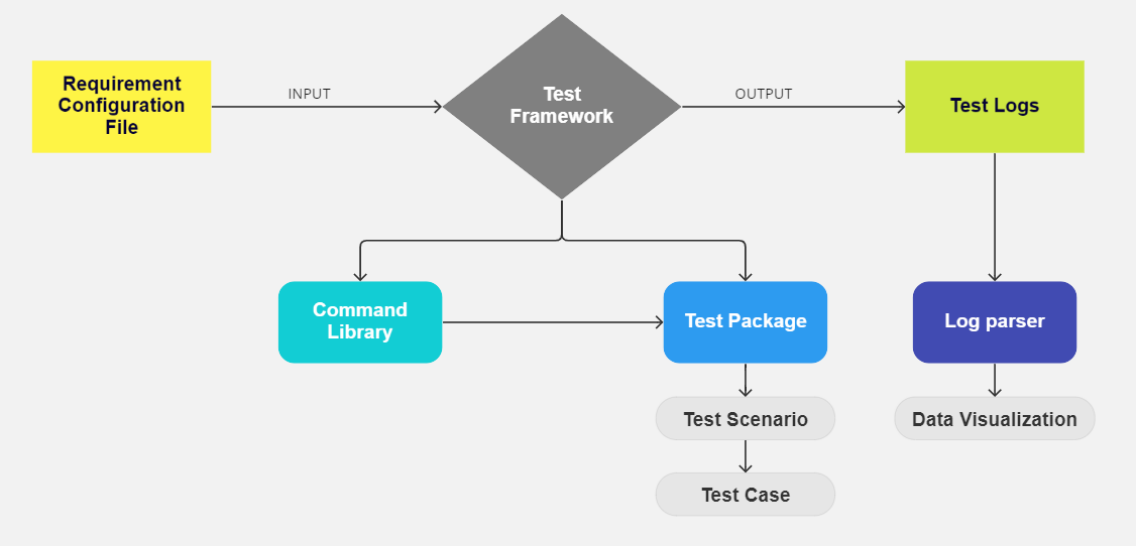


Fig 2.1. Test framework main blocks

2.4. Data visualization

Once the results of the metrics measured are obtained, there is a set of data that must be processed for easy interpretation. Data visualization is important to abstract the complexity of the performance logs and be able to present the data in an attractive way for stakeholders. This section describes our implementation of the data visualization strategy for the test framework.

Defining a good strategy for data visualization is of crucial importance to developers and clients since both find it convenient to access the most amount of data in an intuitive way and be able to make comparisons and observe the evolution of the product.

Several options were considered (Figure 2.2), such as different bar plots showing the average, maximum and minimum times of each command in every sprint.

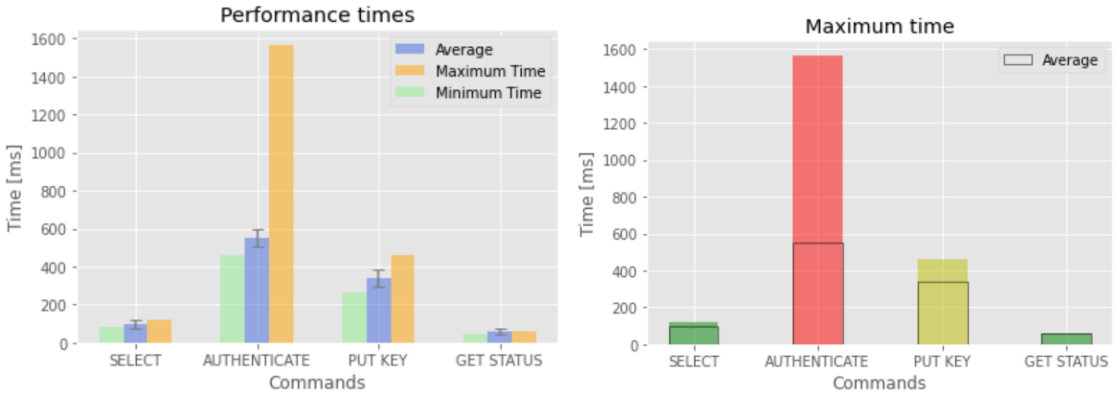


Fig 2.2. Data visualization proposals

It was a requirement for the plot to show the temporal evolution of the commands throughout the different intervals of implementations called Sprints (refer to section 3.2 of this document). It should also contain all the information of the average, deviation, maximum and minimum times measured.

In the end, several options were given to the stakeholders to choose the most attractive and explicative plot. The best approach chosen by the stakeholders was a candlestick (Figure 2.33) plot similar to those widely used in the cryptocurrency market to contain the average time and deviation in the real body of the figure, and the maximum and minimum times in the wicks of the candlestick.

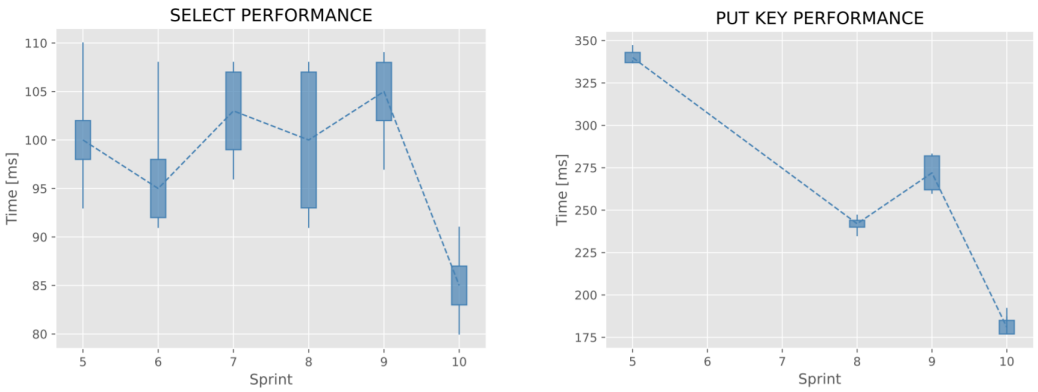


Fig 2.3. Data visualization chosen

The performance logs would contain the final results of all the sprints measured. However, some commands are implemented in the OS later or can be temporarily disabled due to internal revision, bug fixing, or change in requirements. Considering this, the data visualization system must be tolerant to empty data.

Then, the design for the parsing of the logs was to identify the labels and metrics to create a map-like structure in which the keys are the commands and the values are arrays of sprints, each containing, in turn, arrays of 4 positions: the average, standard deviation, minimum and maximum times.

The data of a sprint in which a particular command is not measured will be all null and, thus, checks for null values must be done when plotting. After choosing the type of plot and the design of the parsing, the library matplotlib allows us to draw patches and customize the candlestick to the use case.

Finally, the input to the visualization scripts will be the path where the log file generated after the test can be found, and the output is a pdf file with the plots for every command. The use of this tool is directed at software developers, so the most convenient way to present it is via a script that can be executed from a command line, or automatized in a continuous integration server and, therefore, no graphical interface was developed.

The script for this data visualization tool can be found in the Annex of this document.

2.5. CI/CD

Continuous Integration (CI) is a common practice in software development, which consists of the frequent integration of the code implemented into the main branch.

Branches are used for version control of the code, to keep track of the working software, and to be able to revert changes or perform easier fixes if errors are introduced.

The common practice for CI involves working on a branch that is specific to the new functionality to be added, and once this is tested and is found to be compliant with quality standards, it is merged into the main branch of the product. Integration errors can be detected sooner, and there is always a reliable reference of the latest batch of working tested software in the project.

In the main branch, the code in compiled and executable builds are generated to be tested via automated tests of continuous integration servers.

These servers are configured with trigger tasks to generate the build and run a set of predefined tests; also the logs of the tests can be generated and stored to be checked at any point of the project.

Continuous Delivery (CD) is related to the idea of delivering working software in small batches to have a good measure of progress through functional code that can be tested and even used in some cases.

This is the case of the performance tests developed in this project, which were available for its use before all the functionalities were implemented.

These tests had to go through an exhaustive test of several targets, some of them not considered in the objectives of this version of the product. This means that the tests might not cover the behavior of all products since some have different or specific requirements according to the use cases they are testing.

This would make the targets not meant for these tests to fail, in the remote automatized general executions, and report some errors.

These errors could be traced and a request to exclude these targets was made. Then, the targets not supported won't make the executions fail and these targets will wait until the future tasks of adaptation were done in new iterations of the tests if required.

Continuous Integration (CI) is a common practice in software development, which consists of the frequent integration of the code implemented into the main branch.

Branches are used for version control of the code, to keep track of the working software, and to be able to revert changes or perform easier fixes if errors are introduced.

The common practice for CI involves working on a branch that is specific to the new functionality to be added, and once this is tested and is found to be compliant with quality standards, it is merged into the main branch of the product. Integration errors can be detected sooner, and there is always a reliable reference of the latest batch of working tested software in the project.

In the main branch, the code in compiled and executable builds are generated to be tested via automated tests of continuous integration servers.

These servers are configured with trigger tasks to generate the build and run a set of predefined tests; also the logs of the tests can be generated and stored to be checked at any point of the project.

Continuous Delivery (CD) is related to the idea of delivering working software in small batches to have a good measure of progress through functional code that can be tested and even used in some cases.

This is the case of the performance tests developed in this project, which were available for its use before all the functionalities were implemented.

These tests had to go through an exhaustive test of several targets, some of them not considered in the objectives of this version of the product. This means that the tests might not cover the behavior of all products since some have different or specific requirements according to the use cases they are testing. This would make the targets not meant for these tests to fail, in the remote automatized general executions, and report some errors.

These errors could be traced and a request to exclude these targets was made. Then, the targets not supported won't make the executions fail and these targets will wait until the future tasks of adaptation were done in new iterations of the tests if required.

CHAPTER 3. PROJECT MANAGEMENT

This project was handled as a software development project in a Research & Development department that uses Agile to enhance the efficient development and quality tracking of the product.

These methodologies will be explained in the subsequent sections, along with the planning, timing and management of the project.

3.1. Agile

Agile is an approach to project management of software development that provides a way to deliver value to customers throughout the development process. It allows for products to be more flexible to customer needs and for communication to be fluid inside the project.

The project becomes a cyclic process (Figure 3.1) of development in small batches that involve all the phases of planning, designing, developing, testing, deploying, reviewing, and launching.

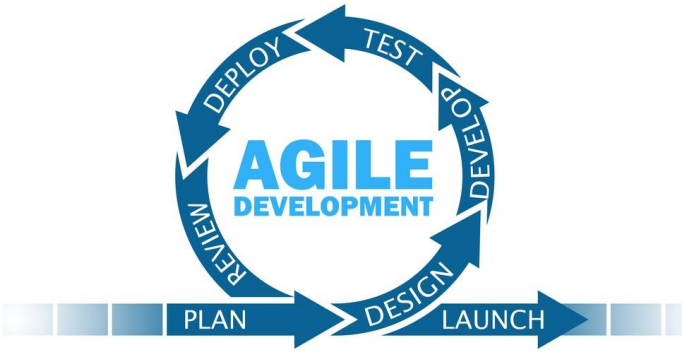


Fig 3.1. Agile cycle or development (see reference [11])

Agile was officially created in 2001 after an alliance of 17 software engineering consultants published the 'Agile Software Development manifesto' (see reference [12]), which stated the principles to manage agile software projects.

The 12 principles of Agile are:

- **Continuous Delivery:** the highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- **Embrace changes:** requirement changes are affordable for agile projects since the planning and development are done in small batches. Changes can be introduced with less impact on the project timeline, which achieves a competitive advantage.

- **Short intervals:** working software delivery must be frequent as agreed with the customer.
- **Daily coordination:** management and developers must work together daily throughout the project.
- **Motivated team:** provide a good environment and support to the team and trust them to self-manage.
- **Face-to-face conversation:** understood as the most efficient way to communicate within a development team.
- **Working software:** as the primary measure of progress and test-driven development, valued higher than documentation.
- **Constant pace:** find a way for the team to make steady progress in a constant manner.
- **Continuous improvement:** attention to technical excellence and quality all along the process.
- **Simplicity:** reduce complex problems to simple tasks so processes become splittable and parallelizable.
- **Self-organization:** teams can be more creative and achieve the best designs and architecture when required to self-organize.
- **Self-improvement:** teams reflect at regular intervals on how to improve and become more efficient.

These principles were applied in the project using a specific agile framework called Scrum.

3.2. Scrum

Scrum is a framework that provides a set of roles, events, and practices that help to create an agile development team (see reference [13]).

Each project is divided into a set of tasks that form the ***Backlog***, which is the list of tasks to solve in the project. This list is dynamic since new requirements or designs might come along at any point of the project.

The backlog tasks get organized in ***Sprints***, which are batches of work estimated to be completed by the team within the span of fixed time intervals. These time intervals are called Sprints; in this project, they were sprints of two weeks each.

The tasks are usually called issues or tickets and they can be tagged as ***Story*** if they imply a new implementation, or '***Defect***' if they imply a fix of an error detected.

These tickets are estimated in Story Points as a measure of the complexity of the task by the whole team. It is important to estimate in terms of complexity and not in time because the comparison of story points and time in a sprint is what can give a metric of the performance of the team.

Hence, a good performance forecast of the team will give a measure of the affordable Story Points in each sprint, so that the work is suited to the team increasing the speed gradually as the team evolves more organically. In this way, sustainable growth is achieved.

3.2.1. Roles

A scrum team consists of three main roles (see reference [13]) as depicted in Figure 3.2:

- **Product Owner:** is responsible for understanding the requirements of the project and creating the backlog with the tickets for every task involved, ordered by priority. This backlog will be updated with any changes in the project. This figure also communicates with the stakeholders and negotiates any changes of requirements and timings in the project.

- **Scrum Master:** is responsible for holding and moderating the Scrum ceremonies. This figure is also a coach for the team defending the scrum procedures and agreed practices for technical excellence, as well as detecting blocking points and helping to enhance effective technical communication.

- **Developers:** they are responsible for the implementation and quality of the tasks reflected in the tickets, they are aware of the backlog and the sprint might suggest the creation of new tasks and modification of existing ones.

Fig 3.2. Agile roles (see reference [14])

All members of a scrum team are capable of creating and valuing each sprint to have an agreement and commitment to the tasks planned; they all attend the Scrum ceremonies to work efficiently, plan together and find improvements for the team.

3.2.2. Ceremonies

The Scrum ceremonies are the events that enable the coordination of the team to work according to the agile methodology, they cover the phases of planning, developing, and conclusion of the sprint as well as sessions of problem-solving to reflect on ways to improve the performance and quality of the team in the long term.

There are 4 main scrum ceremonies for each Sprint (Figure 3.3):

- **Sprint Plan:** this is an event that takes place at the beginning of the sprint when each team member has analyzed some of the tickets in the backlog and proceeds to explain it to the team to estimate it.

  These estimations will help the team agree on the tasks that they can commit to fulfilling in the next sprint based on their performance and availability. In this meeting, it is also agreed on the Definition of Done of the task, which is the procedures and quality testing needed to be done to consider the task concluded and closed.

  Sometimes a previous meeting called Refinement Session takes place in which the team makes sure that each ticket is well defined, so no misunderstandings are caused later on during the sprint.

- **Daily:** these are short daily stand-up meetings in which the whole team updates the status of the ongoing tasks, and possible blocking points or misunderstandings are detected. This is a way to keep daily track of the sprint and be able to be responsive to changes or risks.

- **Sprint Review:** this is the closing ceremony of the sprint. The team reviews the tasks of the sprint both completed and not completed, if any. Then, the team analyses the improving points and the good practices to maintain regarding the tickets of the sprint.

- **Retrospective:** this is an event done with a certain periodicity defined by the team to reflect on how to solve long-term issues, how the performance, and the technical excellence of the team.

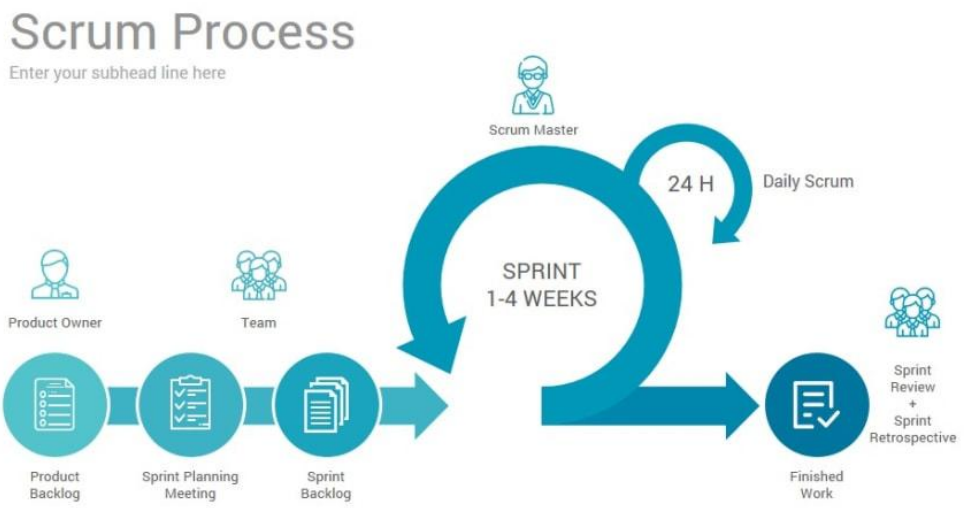


Fig 3.3. Scrum process (see reference [15])

3.3. Roadmap

This project was managed following the Scrum Framework, beginning with the creation of the backlog, supervised by the Product Owner. In order to do this, the requirements and specifications were checked and the different test cases were designed.

Once the commands to be tested were planned and the test cases were designed, the tasks could be created and they could be ordered in terms of priority, also considering the dependencies between tasks.

3.3.1. Pert Chart

In order to remove as many blocking points as possible in the planning phase, a Pert Chart was generated to visualize the dependencies between commands (Figure 3.4).

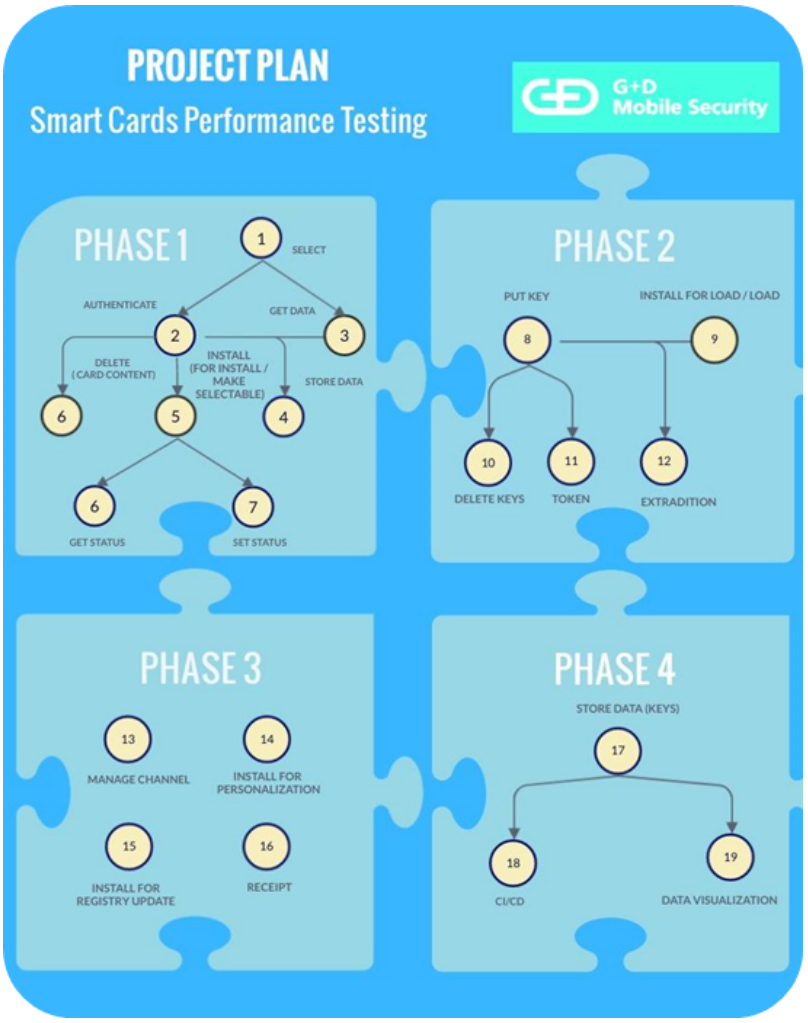


Fig 3.4. Pert Chart of the project

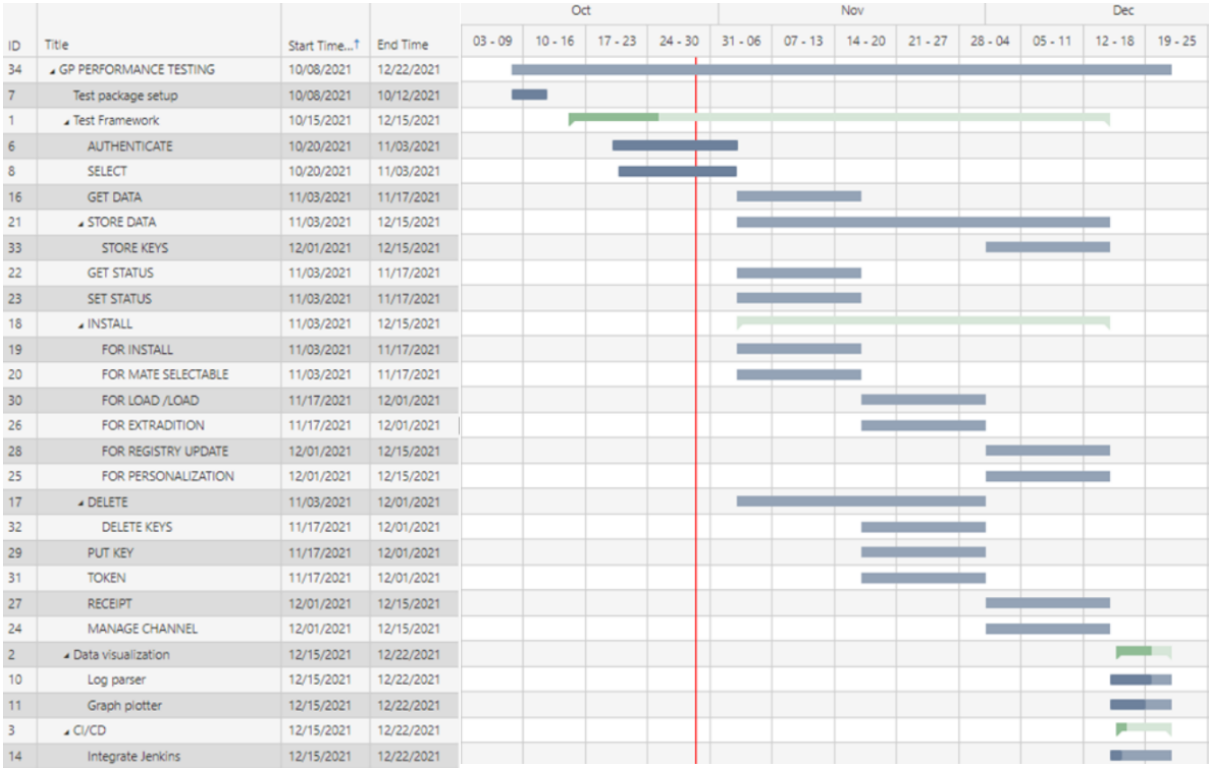
For instance, to test the AUTHENTICATE command (item 2 in the phase 1 of the Pert chart shown in Fig. 3.4), it is required to SELECT an application first. Since the ISD is a pre-installed application in the card, no INSTALL is needed. Hence, it is necessary to implement the SELECT (item 1) first to select the ISD, and then test the AUTHENTICATE, which allows the opening of a secure channel, required to send commands like DELETE, INSTALL and STORE DATA (items 6, 5, and 4 of phase 1, respectively).

Following this analysis, all commands were distributed in the project in 4 phases. The phases are meant to be checking points with stakeholders to check the state of the project.

3.3.1. Gantt Chart

According to the aforementioned distribution of tasks, dependencies, and priorities in the project, a Gantt Chart was created to generate an estimated timeline of the project.

The project would start in the first week of October 2021, and last until the third week of December. Having completed the 4 phases of the project in 6 Sprints of 2 weeks each.



ID	Title	Start Time	End Time
34	GP PERFORMANCE TESTING	10/08/2021	12/22/2021
7	Test package setup	10/08/2021	10/12/2021
1	Test Framework	10/15/2021	12/15/2021
6	AUTHENTICATE	10/20/2021	11/03/2021
8	SELECT	10/20/2021	11/03/2021
16	GET DATA	11/03/2021	11/17/2021
21	STORE DATA	11/03/2021	12/15/2021
33	STORE KEYS	12/01/2021	12/15/2021
22	GET STATUS	11/03/2021	11/17/2021
23	SET STATUS	11/03/2021	11/17/2021
18	INSTALL	11/03/2021	12/15/2021
19	FOR INSTALL	11/03/2021	11/17/2021
20	FOR MATE SELECTABLE	11/03/2021	11/17/2021
30	FOR LOAD /LOAD	11/17/2021	12/01/2021
26	FOR EXTRADITION	11/17/2021	12/01/2021
28	FOR REGISTRY UPDATE	12/01/2021	12/15/2021
25	FOR PERSONALIZATION	12/01/2021	12/15/2021
17	DELETE	11/03/2021	12/01/2021
32	DELETE KEYS	11/17/2021	12/01/2021
29	PUT KEY	11/17/2021	12/01/2021
31	TOKEN	11/17/2021	12/01/2021
27	RECEIPT	12/01/2021	12/15/2021
24	MANAGE CHANNEL	12/01/2021	12/15/2021
2	Data visualization	12/15/2021	12/22/2021
10	Log parser	12/15/2021	12/22/2021
11	Graph plotter	12/15/2021	12/22/2021
3	CI/CD	12/15/2021	12/22/2021
14	Integrate Jenkins	12/15/2021	12/22/2021

Fig 3.5. Gantt Chart of the project

Some tasks like GET STATUS, SET STATUS were planned to go parallelized over a sprint since there was no dependency between them. This is the way that the tasks were later on distributed along with the sprints from the backlog, considering the priority, the timeline, and the dependencies between them.

3.4. Development

After the planning phase was concluded and the roadmap was presented to management, the project began using the agile practices that the scrum framework provides.

Sprint plans were used to estimate the tickets and open the sprint with a set of tasks to achieve. Then, daily meetings were conducted to update on the status of the sprint. In those meetings, the first blocking points and delays started to be spotted.

One of the blocking points was to obtain access rights to all the repositories related to the test framework. There was also a delay caused by underestimating the learning curve of the test framework and the creation of the new package as well as its proper compilation and synchronization with the testing tools.

In this point, the scrum framework provided us with a high degree of traceability and a fast way to notice and inform stakeholders on timing changes.

Through sprint reviews, bottlenecks were identified, such as the long execution times to test the complete package. Then, solutions could be found, for example, getting to the agreement of testing individual test cases manually with a simulation target to find if the commands were parsed correctly, and if all requirements were added.

Then, retrospective sessions worked to find long-term solutions to performance issues. Sometimes it was difficult to drop ideas or designs quickly to find better solutions or to face a change in requirements, these situations had a significant impact on the performance. Embracing the change was a hard task to accomplish, but the retrospective sessions helped to achieve this purpose by creating a new mindset.

The intermediate releases of working code, which is part of agile philosophy, worked perfectly to achieve a better negotiation with stakeholders about the delays in the project. Some blocks of the testing package were already functional while others were being developed.

The functional blocks could already start to be executed by the teams to test the performance impact of their implementations in some of the most priority commands.

When the product started being tested whilst still in the developing phase, some defects could be traced and added to the backlog of tasks. Also, some documentation for training was generated to help the teams use the test.

In the end, the project finished in the last week of February 2022 with the basic functionality covered, and with tasks planned for future versions in which the test will cover more targets, products, and deviations.

The performance tests became a crucial tool for the international teams to cooperate and discuss strategies for the performance optimization of the product. These tests have been executed in the product being developed and other versions in the market to compare each sprint, each integration, and each implementation that could impact the performance.

Through this project, all different Scrum roles were experienced firsthand. Tasks of the Product Owner were carried out, such as creating, organizing, and updating the tickets in the Backlog. Scrum Master duties were also done, such as holding the Scrum ceremonies and following the status of the tickets to identify blocking points. Developer tasks were done all the way to implementing the tickets of the project.

CHAPTER 4. AEROSPACE APPLICATIONS

One of the duties of a Research and Development department is to understand the possible future needs for new products and technological solutions demanded by the market.

In this section, the current state and future opportunities in the market of Smart Cards in Aerospace will be analyzed.

4.1. State-of-the-art of Smart Cards in Aerospace

The applications of smart cards in aeronautics have been limited to ground operations in the form of electronic passports, tickets, accreditations payments, airports blockchain cryptocurrency, and luggage tracking (see reference [3]). Security provided by this secure element with tamper-proof hardware made all these applications possible in a safe way.

Since unmanned air vehicles became available for the general public flying at lower heights, the question arises if there are applications that relate smart cards to on-air operations.

Currently, Smart Cards (SIM cards) are used in some drones to get access to the cellular network to be able to operate by a remote control beyond the line-of-sight (BLOS).

In the late years, the vision of Smart Cities and drones being a key part of them has gained importance, supported by research and stakeholders funds. Drones have the potential of performing emergency and medical missions, surveillance, package delivery, search and rescue operations, infrastructure inspections, and more.

The European Commission intends to promote drone applications as part of this shift in the economy. In this line, there have been Technical Interchange Meetings (see reference [16]) in NASA and ICAO to define a way to manage the increasing amount of Unmanned Aircraft Systems (UAS) that will be flying over the air space at low altitudes.

4.2. Future Applications

The term Unmanned Aerial System Traffic Management (UTM) (Figure 4.1) relates to the realistic strategies to afford safe air traffic control of the new scales of vehicles on air, as well as safety measures on how to distribute the airspace and no-fly zones over the cities. This is an opportunity for Smart Card and Cellular Network providers to offer technological and secure solutions to these new scenarios.

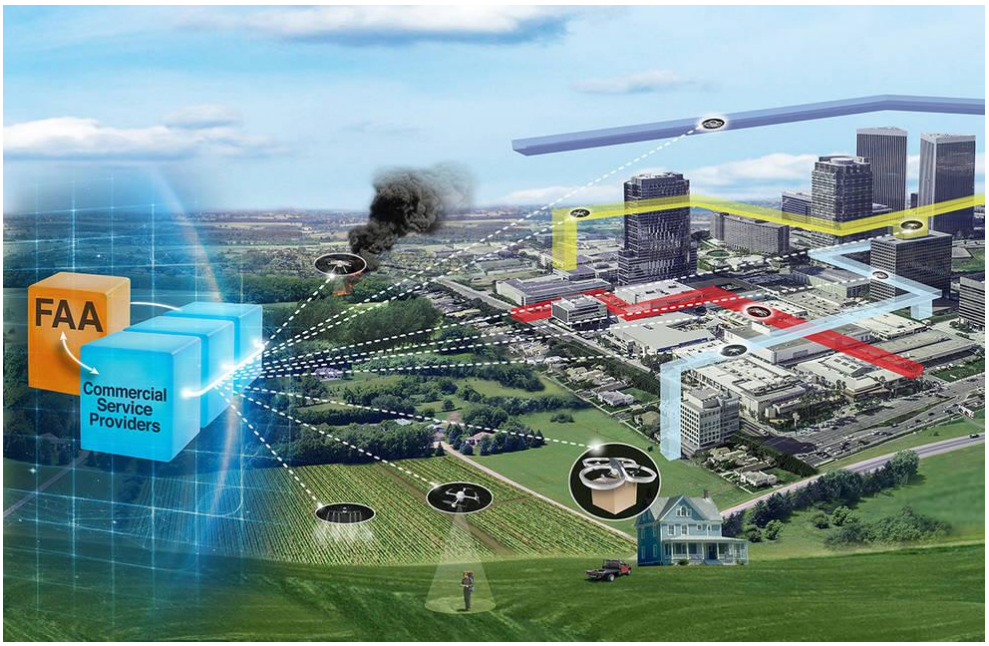


Fig 4.1. UTM scenario introduction scheme (see reference [16])

Drones have been demonstrated to enable very valuable applications for society, such as delivering time-critical medical supplies, evaluating disaster areas, locating lost children, and creating cinematic art (see reference [17]).

To unleash the full potential of these technologies, airspace authorities are working on defining safe traffic management for these new use cases, considering that the growth in the number of vehicles and operations at low altitudes will increase largely over the next years.

NASA's Advanced Air Mobility research is also considering UTM as a baseline for research into air taxis that might coexist with drones at low altitude regions.

UTM is encouraged to become a reality by many sectors and enterprises that believe in the economical growth that this new concept and low-altitude regulated UAS flights can bring. UTM research has had successful field demonstrations and initial operational capabilities have been supported by Fortune 500 companies like Amazon, Alphabet, FedEx, Uber, and UPS. The Urban Air Mobility ecosystem brought by drones can generate a $1.5 trillion market by 2040 (see reference [17]).

The UTM concept created by ICAO and NASA is being adopted worldwide with implementations reported in Europe, India, Singapore, Japan, Australia, and other locations, all of them getting ready to ensure interoperability and safety in their regulations. The systems proposed to involve both visual and BVLOS (Beyond Visual Line of Sight) UAS activities at low altitudes, generally below 400 feet above ground level.

There are strong reasons to believe that new frameworks for UTM will use cellular technology and SIM cards as an identifier and as enablers of secure access.

According to GSMA claims, cellular technology is already part of the drone ecosystem, it is ubiquitous and evolving with well-defined standards for UTM use cases (see reference [18]).

Cellular networks can cover the needs of the systems for UTM being defined in which ATM and Law Enforcement Authorities, apart from the pilot, must have access to the drone and to the UTM framework that will automate some processes such as:

- **Authorization applications:** registration, identification (also done during the flight), flight planning, flight configuration, flight authorization, flight log, playback.

- **Flight support applications:** flight control, geo-locating/tracking, geo-fencing, flight path deviation, remote intervention, airtime monitoring, collision avoidance

- **Data applications:** weather data and airspace information.

Law Enforcement Authorities must be able to control the access to no-fly zones and the unlawful behavior of the aircraft. ATM must have access to remotely manage the routes safely and give flight support. The pilot must have access to the normal operation of the drone.

Mobile cellular networks can benefit from their existing infrastructure to provide low-cost solutions and ubiquitous service. Their secure communication channels provide specific encryption mechanisms to protect communications and keep good levels of data protection and privacy. The use of a licensed band can provide reliable connectivity for mission-critical applications, such as BVLOS cases, and in high-risk environments.

Furthermore, the future of cellular networks with 5G will provide:

- **Higher bandwidth:** enhanced payload data transmission capabilities, such as high-resolution video.

- **Lower latency:** faster command and control (C2) link and detection and avoidance can be commanded by off-board data sources.

All of this matches with scenarios proposed in which drones have connections between them for collision avoidance and with the UTM framework as well as the law enforcement authorities. The base stations can also serve and help trace the no-fly zones and collision avoidance limitations (Figure 4.2).

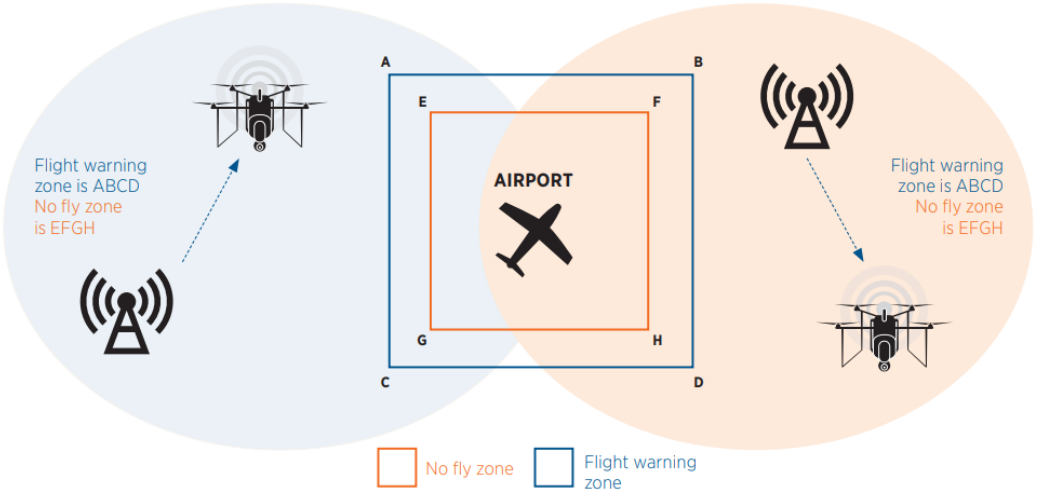


Fig 4.2. UTM scenario for collision avoidance with base stations (see reference [18])

Hence, in this scenario, smart cards play a key role in the implementation of UTM via cellular networks.

SIM cards provide a secure way to link the device to a physical person or legal entity subject to a license and unique identifier. Hardware unique worldwide identifiers like IMEI and IMSI can be used to identify the UAV.

Secure session encapsulation can provide extra safety since the location of origin and destination could be masked from any possible malicious eavesdrop, using a technology similar to onion networks of several encryption levels used in the deep web.

Besides the obvious benefits of enabling cellular-based Internet connection for user-level data exchange between the drone and any external elements, this technology provides identification, authentication, secure connection, and positioning along with the cellular network and a C2 link to remotely and forcefully control the aircraft from the ATM and Law Enforcement authorities, if needed (see reference [18]). The connection between vehicles can also be established to perform collision avoidance maneuvers (Figure 4.3).

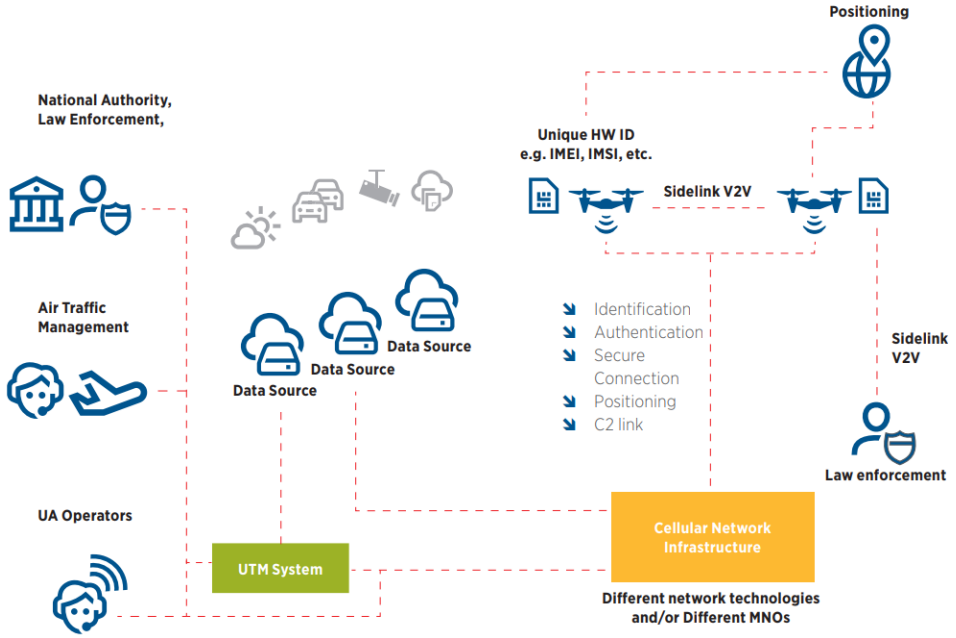


Fig 4.3. UTM scenario with cellular networks and SIM cards. (see reference [18])

The future holds endless possibilities after UTM Systems are fully implemented with cellular networks. New applications might become a reality such as autonomous flight using technologies like Artificial Intelligence and Cloud Computing for route optimization.

Cities would then have different air transportation technologies co-existing seamlessly while making use of different new dedicated aeronautical infrastructures (Figure 4.4).

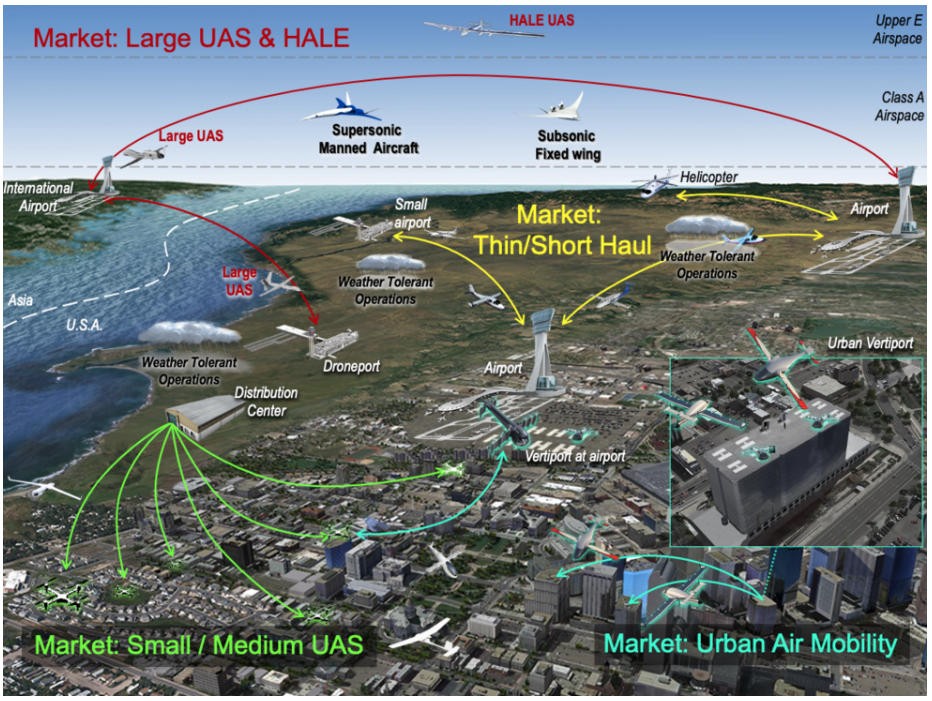


Fig 4.4. UTM scenario of aircraft in urban areas (see reference [19])

Droneports outside of urban areas would be used as hubs for operations of large UAS and then urban vertiports over the cities to be short-time destinations for operations of smaller drones or air taxis that provide services to the city.

Distribution centers are planned to hold the operations of small and medium UAS, which will be allowed to land in urban vertiports or dedicated vertiports inside conventional airports.

4.3. Smart Card features to enhance UTM

Once the future scenarios for aerospace applications of smart cards are analyzed, a question arises about the need to offer new or enhanced smart card products, customized to the requirements of these new applications.

Some features can be identified as necessary when considering the scenarios previously presented. In order to offer full access and control to pilot, ATM, and Law Enforcement Authorities, the multi-profile features of smart cards will play a key role since there can be different containers holding the different profiles, their own architecture as explained in Chapter 1 of security domains, applications, and sensitive data well differentiated from each profile.

The Subscription Management (SubMan) functionality allows several Mobile Network Operators (MNOs) to have a profile in the card as depicted in Figure 4.5. In this way, the connectivity would be enhanced by being able to subscribe the device to all MNOs needed to serve the route as well as ATM and Law Enforcement Authorities.

The SubMan functionality is designed by the Original Equipment Manufacturer (OEM) which can define levels of priority that could be established between profiles to offer a hierarchy in which Law Enforcement and ATM authorities will get prioritized access in case it is needed.

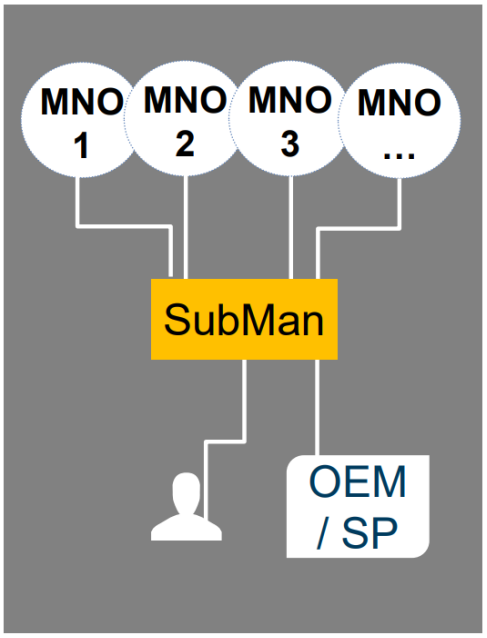


Fig 4.5. SubMan scenario (see reference [20])

Logical channels could also be reserved and prioritized, for example, in the case of emergencies and contingency measures. Then, the card can guarantee that commands related to an emergency will be processed first.

New encryption mechanisms can be held to secure communications with highly secure and efficient hardware-embedded cryptographic engines.

These are some of the many features that can be involved in the requirements of the new aerospace applications of Smart Cards. Market studies for feature and product development would take place in order to launch well-defined projects to approach these new opportunities.

However, these features would also need some adaptations of current UAV communication protocols like MavLink (see reference [21]). This protocol includes support for all sorts of commands for flight control and telemetry data sharing so the ground control systems can communicate with the flight controller of the UAV and control it.

MavLink already has commands related to interaction with the SIM card known as “CELLULAR_CONFIG” and “CELLULAR_NETWORK”, hence in this set of commands some updates could be made to involve SubMan capabilities regarding ATM and Law Enforcement Authorities. The pilot commands can be rejected while the ATM or Law Enforcement Authorities profile is selected.

These changes will indeed be subject to certifications and this is why the whole industry develops together and standards are reviewed by stakeholders to find a unified way to offer technological solutions to these scenarios.

CHAPTER 5. CONCLUSIONS

This project defined a journey going through all the stages of product development in a R&D department. Through this journey, we could get an insight on the specification and test-driven approach to products to meet high-quality standards and comply with certifications.

From the planning phase, a deep analysis of specifications, standards, and documentation is done to be able to picture the whole image of the project, its requirements, constraints, and risks.

Then, project management methodologies apply to ensure a good relationship with stakeholders since the beginning of the project, and software dedicated tools for efficient development of software are applied.

We learnt that traceability and effective communication are key factors in the success of a project since change and continuous improvement should be embraced in this environment.

The importance of test-driven developments was also proved, as well as the impact of running performance tests along the development. It enhances decision-making based on metrics and not opinions, always focused on the product goal and the maximum quality.

Through the development of the tests, expertise with the test framework and test-driven development methods was achieved. The standards were reviewed to make sure that the behavior of the tests matched the theoretical behavior of the product that will be certified. A large amount of coverage was obtained due to the analysis of the particular test cases as well as the proposal of real-life scenarios.

We could also learn the methodologies that are used to make interdisciplinary teams work towards common objectives, in a unified way with communication and constant feedback to achieve technical excellence.

The continuous delivery of small batches of viable products reduces the risk of creating new projects. The quality and functional testing can track the viability of these projects from the start and discover early any technological barriers or bottlenecks that might make the project decline with the low risk taken and low budget invested.

In this way, the traceability and testing enhance once again the ease to pose creative solutions to the needs of the market without large budgets needed since the product is divided into versions and viability can be tracked in each step of the process.

A good insight into the product development process can make stakeholders feel a bigger sense of control over the projects and a lesser risk of the investments made.

Finally, we discussed on the applications of Smart Cards in Aerospace and the future opportunities that drones and smart cities provide to this industry. Cellular networks along with smart card technology will provide a solid secure background for the new applications that will transform the cities and the future of air transportation.

Cities will be transformed as well as aeronautical infrastructures with the tentative designs of droneports and vertiports. An insight of a future in which UAS and conventional aircraft can co-exist in a seamlessly secure environment.

Then, after analyzing the implications and the impact of these future applications, we can see a new market opening for new smart card products of versions which takes us to the starting point of the process.

A never-ending process of identifying the needs of the market, customizing products to meet the requirements, and then planning, developing, testing, certifying, and launching. Creating today the product for the future.

The objectives of this project were achieved successfully since the project concluded with a set of functional performance tests that are being used by international teams developing new products. Some documentation and training were done for the teams to use the tests on the daily basis of their implementations and decision-taking, also as a way of damage control.

Project management methodologies like Scrum were experienced firsthand and built a solid background to become a reference for the team.

Finally, the research for aerospace applications will open market opportunities for our products and set the future lines for this project which will be to follow the UTM research closely to plan for new products and new features that will be developed and tested with our test framework.

These performance tests will be extended to cover future use cases, features, and requirements. These tests will keep on being used to enhance the efficient development of quality secure products.

REFERENCES

[1] BNP Paribas. “Smart Cards, A French Invention That Revolutionised Payments (1/2).” https://histoire.bnpparibas/en/smart-cards-a-french-invention-that-revolutionised-payments-12-before-the-chip/.

[2] BNP Paribas. “Smart Cards, A French Invention That Revolutionised Payments (2/2).” https://histoire.bnpparibas/en/smart-cards-a-french-invention-that-revolutionised-payments-22-across-the-world/.

[3] Thales. “Smart Cards - A short Review (Feb. 2022).” 8 February 2022, https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/smart-cards-basics.

[4] Eurosmart. “The voice of the Digital Security Industry | 2019 shipments and 2020 outlook.” 10 June 2020, https://www.eurosmart.com/2019-shipments-and-2020-outlook/

[5] Pawlak, Adam. “Typical smart card architecture. Scientific Diagram.” *ResearchGate*, https://www.researchgate.net/figure/Typical-smart-card-architecture_fig2_227074206.

[6] Kaspersky IT Encyclopedia. “What is Secure Element?” https://encyclopedia.kaspersky.com/glossary/secure-element/.

[7] Bennett, Jane. “Smart Card. Smart Card applications.” 18 April 2016, https://silo.tips/download/smart-card-smart-card-applications.

[8] GlobalPlatform. “Card Specification v2.3.1.” March 2018, https://globalplatform.org/wp-content/uploads/2018/05/GPC_CardSpecification_v2.3.1_PublicRelease_CC.pdf.

[9] Sakane, Hirofumi, and Caroline Scace. “FIPS 140-3 Non-Invasive Attack Testing”, https://csrc.nist.gov/CSRC/media/Presentations/FIPS-140-3-Non-Invasive-Attack-Testing-Presentatio/images-media/noninvasive-attack-testing_cscace-hsakane.pdf.

[10] International Standard Organization and International Electrotechnical Commission. “International Standard ISO/IEC 15408-1.” 15 December 2009, https://webstore.iec.ch/preview/info_isoiec15408-1%7Bed3.0%7Den.pdf.

[11] D'Ambra, Scott. “What is Agile Software Development?” *ClearTech Interactive*, https://www.cleart.com/what-is-agile-software-development.html.

[12] S. Alsaqqa, S. Sawalha, and H. Abdel-Nabi, “Agile Software Development: Methodologies and Trends”, *Int. J. Interact. Mob. Technol.*, vol. 14, no. 11, pp. pp. 246–270, July 2020, https://doi.org/10.3991/ijim.v14i11.13269.

[13] K. Schwaber, and J. Sutherland. “The Definitive Guide to Scrum: The Rules of the Game.” *Bill Lewis Training*, https://billlewistraining.com/wp-content/uploads/2017/02/PMP-Agile-Study-Materials.pdf.

[14] Balasegu. “Scrum and Scrum teams.” *One Stop for Testing and Tools*, https://balasegu.weebly.com/scrum-and-scrum-teams.html.

[15] Euda. “Improve your performance with Scrum Methodology - DEV Community.” *DEV Community* , 24 January 2020, https://dev.to/euda_ar/improve-your-performance-with-scrum-methodology-idg.

[16] NASA Aeronautics Research Institute. “UTM Project Technical Interchange Meeting (TIM).” 23 February 2021, https://nari.arc.nasa.gov/utm2021tim.

[17] NASA. “UTM 101.” 26 June 2020, https://www.nasa.gov/aeroresearch/utm-101/.

[18] GSMA. “Using Mobile Networks to Coordinate Unmanned Aircraft Traffic.” *GSMA*, 2018, https://www.gsma.com/iot/wp-content/uploads/2018/11/Mobile-Networks-enabling-UTM-v5NG.pdf.

[19] Urban Air Mobility Noise: Current Practice, Gaps, and Recommendations - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Integration-of-UAM-into-airspace_fig1_344493775.

[20] GSMA, and Giesecke & Devrient. “Benefits of Remote SIM Provisioning - MWC Shanghai 2015.” *GSMA*, 15 July 2015, https://www.gsma.com/iot/wp-content/uploads/2015/07/009_Klaus-Vedder_EN.pdf.

[21] MAVLink. “MAVLink Developer Guide.” https://mavlink.io/en/.

ANNEX

Smart Card Performance Testing Tool

Data visualization for test logs

María Teresa Fernández Mateos, January 2022

Read the log containing the summary of the metrics and plot the performance of each command found in the log and save a PDF file with the plots

Open and read the log file in the given path:

```python
# Input the path where the log file named "Log.txt" will be placed
print("Enter the path of the test log file:")
filename = input() + "\Log.txt"

#Try to read the file
try:
    print(filename, "r")
    with open(filename) as f:
        file = f.read()
        print(file)
except:
    # Catch the error when the log file is not found
    print("Could not find Log.txt in this path.")
```

Python

```
Enter the path of the test log file:
C:\Users\maife\Desktop\TFG\DataVisualization\Log.txt r
*******************************************************
               PERFORMANCE TIME METRICS
*******************************************************
Initial Sprint: 5
Sprints in log: 6
-------------------------------------------------------
SELECT: 100ms +/- 2ms | [93ms, 110ms]
AUTHENTICATE_PRODCUTION: 550ms +/- 4ms | [546ms, 557ms]
PUT_KEY: 340ms +/- 3ms | [338ms, 347ms]
.......................................................
-------------------------------------------------------
SELECT: 95ms +/- 3ms | [91ms, 108ms]
.......................................................
-------------------------------------------------------
SELECT: 103ms +/- 4ms | [96ms, 108ms]
GET_STATUS: 55ms +/- 10ms | [41ms, 68ms]
.......................................................
-------------------------------------------------------
SELECT: 100ms +/- 7ms | [91ms, 108ms]
PUT_KEY: 242ms +/- 2ms | [235ms, 247ms]
GET_STATUS: 50ms +/- 8ms | [40ms, 61ms]
.......................................................
-------------------------------------------------------
...
PUT_KEY: 181ms +/- 4ms | [187ms, 192ms]
GET_STATUS: 49ms +/- 8ms | [33ms, 61ms]
.......................................................
```

Parse the logs and store the results for processing:

```python
# Open the log file to parse it
with open(filename) as f:
    # List containing lines of file
    lines = f.readlines()
    try:
        # Map to store the metrics for each command measured
        metrics = {}
        # List to store the number of each sprint measured
        sprints = []
        # Boolean to indicate when a section of metrics appears in the log
        parse_metrics = False
        # Total number of sprints recorded in the log
        number_sprints = 0
        # First sprint found in the log
        initial_sprint = 0
        # Curent paring sprint
        sprint = 0

        for line in lines:

            # Found the ending of a sprint metrics section
            if line.find(".....................") != -1:
                # Save the parsed metrics
                sprints.append(str(sprint))
                # Update the current sprint
                sprint = sprint + 1
                # Stop parsing metrics
                parse_metrics = False
```

```python
            # Found the initial sprint data
            if line.find("Initial Sprint:") != -1:
                contents = line.split(':')
                # Save the initial sprint
                initial_sprint = int(contents[1].strip())
                # Set the current sprint to the initial sprint value
                sprint = initial_sprint

            # Found the number of sprints data
            if line.find("Sprints in log:") != -1:
                contents = line.split(':')
                # Save the number of srpints measured
                number_sprints = int(contents[1].strip())

            # Found metrics parsing section
            if parse_metrics == True:
                contents = line.split(': ')
                # Correct format
                command = contents[0].replace('_', ' ')
                # Find the average
                contents = contents[1].split('ms +/- ')
                # Save the average time
                average = int(contents[0])
                # Find the deviation
                contents = contents[1].split('ms | [')
                # Save the deviation
                deviation = int(contents[0])
                # Find the minimum time
                contents = contents[1].split('ms, ')
                # Save the minimum time
                minimum = int(contents[0])
                # Find the maximum time
                contents = contents[1].split('ms]')
                # Save the maximum time
                maximum = int(contents[0])

                # If no metrics have been parsed for this command before
                if metrics.get(command) == None:
                    # Initialize the command with null values for all sprints
                    metrics[command] = [None] * number_sprints
                # Fill in the command metr (variable) initial_sprint: int
                metrics[command][sprint - initial_sprint] = [average, deviation, minimum, maximum]

            if line.find("----------------") != -1:
                parse_metrics = True
    except:
        # Catch format errors
        print("Could not read the log. Revise the format.")
        input()

    # Print results of the parsing
    print(sprints)
    print(metrics)
```

Python

```
['5', '6', '7', '8', '9', '10']
{'SELECT': [[100, 2, 93, 110], [95, 3, 91, 108], [103, 4, 96, 108], [100, 7, 91, 108], [105, 3, 97, 109], [85, 2, 80, 91]], 'AUTHENTICATE PRODCUTION': [[550, 4, 546, 557], None, None, None, None, None], 'PUT KEY': [[340, 3, 338, 347], None, None, [242, 2, 235, 247], [272, 10, 260, 283], [181, 4, 187, 192]], 'GET STATUS': [None, None, [55, 10, 41, 68], [50, 8, 40, 61], [50, 6, 41, 59], [49, 8, 33, 61]]}
```

Obtain plots for each command found in the logs and save a PDF with the performance plots.

```python
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.backends.backend_pdf import PdfPages

# Positions of the different metrics in the array generated
average = 0
deviation = 1
minimum = 2
maximum = 3

# Format setting for the plot
color = 'steelBlue'
candle_width = 0.1
candle_linewidth = 1
plt.style.use('ggplot')
figure, axis = plt.subplots()

# Initialize output file
pdf = PdfPages('Performance.pdf')

# For each command measured
for command, time in metrics.items():
    # Set figure and axis
    figure, axis = plt.subplots()
    length_data = len(time)
    # List for the y-axis data
    averages_plot = []
    # List for the x-axis data
    sprints_plot = []
    # Index counter initialization
    index = 0
```

```
    # For each sprint measure found in the commands measures
    for index in range(length_data):
        # If the metric is not null
        if time[index] != None:
            x = index + 1

            # Save the average in the x-axis list
            averages_plot.append(time[index][average])
            # Save the initial in the x-axis list
            sprints_plot.append(int(sprints[index]) - initial_sprint + 1)

            # Draw the rectangle patch of the candle indicating the deviation values
            border = patches.Rectangle((x - candle_width, time[index][average] - time[index][deviation]), 2*candle_width, 2*time[index][deviation], linewidth=candle_linewidth, edgecolor=color, facecolor='none')
            inside = patches.Rectangle((x - candle_width, time[index][average] - time[index][deviation]), 2*candle_width, 2*time[index][deviation], edgecolor='none', facecolor=color, alpha=0.7)

            # Add patch to the plot
            axis.add_patch(border)
            axis.add_patch(inside)

            # Draw the sticks of the candle indicating the maximum and minimum values
            plt.plot((x, x), (time[index][average] + time[index][deviation], time[index][maximum]), color=color, linewidth=candle_linewidth)
            plt.plot((x, x), (time[index][average] - time[index][deviation], time[index][minimum]), color=color, linewidth=candle_linewidth)

    # Draw the evolution of average times
    plt.plot(sprints_plot, averages_plot, linestyle= '--', color=color, linewidth=1.2*candle_linewidth)
    # Empty the lists used for the command plot
    averages_plot = []
    sprints_plot = []

    # Add title and axis names
    plt.title(command + " PERFORMANCE")
    plt.xlabel('Sprint')
    plt.ylabel('Time [ms]')
    plt.xticks(range(1, len(sprints) + 1), sprints)

    #plt.show()
    pdf.savefig(figure)

pdf.close()
```

Python