Software Engineering 2 Project: PowerEnJoy
Project Management Document

- Alessio Mongelluzzo
- Michele Ferri
- Mattia Maffioli

Contents

1 Function Points estimation

Function Points Analysis is a method aimed at estimating software size, widely accepted as an industry standard for functional sizing.

A Function Point (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user.

Five user function types should be identified, as defined in the following subsections of this chapter, which are:

- Internal Logic Files
- External Interface Files
- External Inputs
- External Outputs
- External Inquiries

Measuring criteria for these function types are taken from the COCOMO II Model Definition Manual.[1]

Function Point measurement is done considering Record Element Types (RETs) and Data Element Types (DETs) as defined in the *Function Point Counting Practices Manual* from the International Function Point Users Group (IFPUG).[2]

1.1 Internal Logic Files

An Internal Logic File (ILF) is defined as a homogeneous set of data used and managed by the application.

1.1.1 Measuring criteria

	Data Elements		
Record Elements	**1-19**	**20-50**	**51+**
1	Low	Low	Avg.
2 - 5	Low	Avg.	High
6+	Avg.	High	High

Complexity	**FPs**
Low	7
Avg.	10
High	15

[1] http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf

[2] http://www.ifpug.org/

1.1.2 Complexity analysis

In order to measure the FPs for a specific ILF, we need to analyze the structure of the data contained in it. The ILFs of our system match closely the database tables and are the following:

- **User.** This file contains data about registered users interacting with the system. For each user we can think of the data as structured into 4 RETs, which are: the basic personal details (name, surname, date of birth...), login credentials (email, username, password), address (street, house number, city, state, postcode, country), and billing information (e.g. credit card number, expiry date, CVV, cardholder name and surname). Given that the DETs are 20 or more, we assume the complexity of this ILF is average.

- **Reservation**. This file contains data about car reservations, made by users, which are currently active. The only RET present in this file is the reservation data itself, which is composed of few DETs: date and time, reserved car, and a reference to the user who reserved it. Therefore, we assume the complexity of this ILF is low.

- **Ride**. This file contains information about active and completed rides. For each ride, we can think of the data as structured into 2 RETs, which are: the core information of the ride (driver, car, ride state, begin/end date, payment outcome) and the additional information needed to possibly compute fare modifiers (number of travellers, begin/end position, money saving destination, final distance from charging area, final battery level, the possible discount, and one or more possible additional charges). Given that the DETs are 20 or more, we assume the complexity of this ILF is average.

- **Car**. This file contains information about cars. The only RET present in this file is the car data itself, which is composed of few DETs: license plate, locked, current position, battery level, car state, plugged in. Therefore, we assume the complexity of this ILF is low.

- **SafeArea and ChargingArea**. This file contains information about safe areas. There are two types of safe areas: normal safe areas and charging areas. Each safe area is described by an identifier and the vertices of the polygon which delimits it. Furthermore, recharging stations also need to have the number of plugs specified. This accounts for 3 Data Elements, so this ILF should be considered of low complexity.

- **Employee**. This file contains data about employees working for PowerEnJoy. For each employee we can think of the data as structured into 4 RETs, which are: basic personal details (name, surname, date of birth...), login credentials (email, username, password), address (street, house number, city, state, postcode, country), and the employee's core information

(current state, and possibly a car which the employee is taking care of). Given that the DETs are less than 20, we assume the complexity of this ILF is low.

ILF	Complexity	FPs
User	Average	10
Reservation	Low	7
Ride	Average	10
Car	Low	7
SafeArea	Low	7
Employee	Low	7
TOTAL		**48**

1.2 External Interface Files

An External Interface File (EIF) is defined as a homogeneous set of data used by the application but generated and maintained by other applications.

1.2.1 Measuring criteria

	Data Elements		
Record Elements	**1-19**	**20-50**	**51+**
1	Low	Low	Avg.
2 - 5	Low	Avg.	High
6+	Avg.	High	High

Complexity	FPs
Low	5
Avg.	7
High	10

1.2.2 Complexity analysis

- **Payment System:** all the information the system needs to manage in order to perform payment transactions. The functionalities involved in such operations are mainly 3: initial validity check for billing information (e-mail address, password) provided by the user at the registration, the actual transaction operations with the amount to pay and the outcome the external system has to provide to determine whether a user must be banned or not. Therefore 3 Record Elements and 4 data elements are involved, so this EIF has a low complexity.

- **Google Maps retrieval system:** maps are retrieved specifying a couple of float numbers (latitude and longitude) which corresponds the user position, i.e. the center of the map, and a parameter to determine how far from the user the map should be extended. Two record elements and 3 data elements are involved, so this EIF is again of low complexity.

- **Driving Licenses interface:** in order to check whether the user actually owns a valid driving license, the system sends the user credentials (name, surname, date of birth, address) and the license number to the validation system. Two record elements and 5 data elements are used, therefore it is a low complexity EIF.

EIF	Complexity	FPs
Payment System	Low	5
Google Maps retrieval system	Low	5
Driving Licenses external interface	Low	5
TOTAL		**15**

1.3 External Inputs

An External Input (EI) is defined as an elementary operation to elaborate data coming from the external environment.

1.3.1 Measuring criteria

	Data Elements		
Record Elements	**1-4**	**5-15**	**16+**
0 - 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
4+	Avg.	High	High

Complexity	FPs
Low	3
Avg.	4
High	6

1.3.2 Complexity analysis

- **Registration:** when a user wants to register to the system he/she must provide the following data: e-mail, name, surname, date of birth, address, username, password, billing information, license number. The register operation modifies the User ILF and interacts with the Payment System EIF and Driving Licenses EIF. This EI complexity must be considered average.

- **Login:** in order to perform a login the user needs to provide only 2 data elements (username and password). This operation only interacts with the User entity, i.e. the User ILF. Therefore, we can state this EI complexity to be low.

- **Edit profile:** considering the worst case possible of profile editing, a user could modify every information he/she provided at registration, except for the e-mail and username (name, surname, date of birth, address, password,

billing information, license number). In case of validation of the modified information, this operation would involve the User ILF and the EIF, which are the Payment System and the Driving Liceses interface. This EI can be considered of average complexity.

- **Reserve a car:** the reservation of a car requires the user to query the system for available cars around a position, therefore it must involve the Google Maps retrieval system EIF. To perform the reservation 2 data elements are necessary (user identifier, car identifier). An interaction with the User ILF, Reservation ILF and Car ILF is necessary. As we stated in the Design Document, the user is immediately charged with the reservation deposit as he/she reserves a car, therefore this operation interacts with the Payment System EIF as well. Dealing with 2 data elements and 5 record elements this activity can be claimed to have an average complexity.

- **Begin a ride:** by the time the user turns on the car within the reservation deadline (1 hour), he is immediately refunded with the deposit. The Payment System EIF must then be involved in this operation together with the User ILF. A new tuple is inserted into the Ride entity, i.e. the Ride ILF and the Car ILF is updated. To perform this operation the information about the corresponding reservation is necessary (Reservation ILF). The data elements necessary to create a ride are the ones provided by the corresponding reservation (User and Car) together with the car position. In case the money-saving option is enabled an interaction with the Google Maps retrieval system is necessary. We can once again state this EI complexity to be average.

- **Complete a ride:** once the car is switched off, the Car ILF is updated, and the User is charged with the ride price. User ILF and Payment System EIF are involved. The car position and battery level are the only data elements required. In order to determine whether a fare modifier can be applied, an interaction with the Safe Area ILF and Charging Area ILF is necessary. In case of low battery car the Employee ILF is involved. This EI complexity is assessed to be average.

- **Employee car fix:** once an employee intervention is completed, he updates the Car ILF with the new data: position, battery level. Of course the Employee ILF is involved in this operation. This EI has a low complexity.

- **Edit Safe Areas:** In order to insert a safe area a set of coordinates have to be provided to update the Safe Area ILF. This simple activity has a low complexity.

- **Edit Cars:** the insertion of a new car into the database requires the Car ILF to be updated. The data necessary to perform a valid car insertion are a license plate, the battery level and the position. This EI has a low complexity too.

EI	Complexity	FPs
Registration	Average	4
Login	Low	3
Edit profile	Average	4
Reserve a car	Average	4
Begin a ride	Average	4
Complete a ride	Average	4
Employee car fix	Low	3
Edit Safe Areas	Low	3
Edit Cars	Low	3
TOTAL		**32**

1.4 External Outputs

An External Output (EO) is defined as an elementary operation that generates data for the external environment. It usually includes the elaboration of data from logic files.

1.4.1 Measuring criteria

	Data Elements		
Record Elements	**1-5**	**6-19**	**20+**
0 - 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
4+	Avg.	High	High

Complexity	FPs
Low	4
Avg.	5
High	7

1.4.2 Complexity analysis

- **Unlock message for Ride:** this message is sent by the system to a reserved car, in order to unlock its doors and let its user start the ride as soon as he turns on the engine. Sending this message only requires knowing what car the user reserved, an information that is contained in a Reservation tuple. Hence, interacting with only 1 ILF, this EO has a low complexity.

- **Lock message for Ride:** this message is the opposite of the **Unlock message** and is then sent so that car doors are locked at the end of the ride. This message involves the Car ILF, in order to check which car the message has to be sent to, based on the position. Therefore, also this EO can be considered of low complexity.

- **Unlock message for Employee:** this message is sent by the system to a reserved car, in order to unlock its doors and let the employee start mantaining it. Sending this message only requires knowing the car associated to the employee, which is an information contained in the Employee ILF. Hence, interacting with only 1 ILF, this EO has a low complexity.

- **Lock message for Employee:** this message is the opposite of the **Unlock message** and is then sent in order to lock the car doors after the maintainance is finished. This message involves again the Employee ILF. As the previous one, this EO can be considered of low complexity.

- **Current fare:** this output is computed in the car side and showed in its monitor during a ride. The only information needed for this is contained in the Ride ILF (the begin_date_time attribute), which is shared with the car only to notify the user about the fare (while the final price is based on the information that is in the system, for security reason). Besides, since it only needs one data element, the complexity of this EO is low.

- **Payment:** this output is sent to the Third Part Payment System and requires billing_information (found in User ILF) and begin_date_time and end_date_time of the Ride ILF. These are not sufficient, as the final price is decided upon a list of discounts/additional charges computed after values sent by the car sensors. This has to be taken into account, since, for the additional charge, the distance from the nearest Charging Area (hence another ILF) has to be considered. Furthermore, the payment requires the interaction with the Third Party Payment System EIF. In conclusion, the complexity of this output can be considered average.

- **Maintainance Request message:** this output message is sent to an employee in order to request the maintainance of a car. In this case it needs to interact with 2 ILFs (the Car tuple and the Employee ones, in order to compute their distance, based on their position), so that the message is sent to the employee that is nearest to the car. Hence we can consider it of low complexity.

EO	**Complexity**	**FPs**
Unlock message for Ride	Low	4
Lock message for Ride	Low	4
Unlock message for Employee	Low	4
Lock message for Employee	Low	4
Current fare	Low	4
Payment	Average	5
Maintainance Request message	Low	4
TOTAL		**29**

1.5 External Inquiries

An External Inquiry (EQ) is defined as an elementary operation that involves input and output, without significant elaboration of data from logic files.

1.5.1 Measuring criteria

	Data Elements		
Record Elements	**1-5**	**6-19**	**20+**
0 - 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
4+	Avg.	High	High

Complexity	**FPs**
Low	4
Avg.	5
High	7

1.5.2 Complexity analysis

- **Car lookup:** this inquiry requires an address string, which is translated into a couple of coordinates (that determine the center of the map), and a float (to specify the maximum distance from the address), both received as input from the user. Besides, positions of nearby cars (couples of float) are used to signal them on the map, along with their plate and their battery charge level in order to provide also information about cars in output to the users. So, the inquiry deals with 3 data elements, 1 ILF (because of the car tuples) and 1 EIF (in order to retrieve the map from the Google Maps Retrieval System) and therefore we consider it is of average complexity.

- **Billing information checking:** this inquiry deals with the checking of the billing information provided by users during registration: it interacts with a Third Party Payment System EIF and it deals with the billing information of the user (email, password or in alternative name, surname, address, date_of_birth, credit_car_number, CVV). The only output it provides is whether the billing information is correct or not. It can deal with 6 data elements, but the ILF is only 1, so this inquiry is considered of low complexity.

- **Driving licenses checking:** this inquiry is very similar to the previous one (**Billing information checking**); it requires as user's provided inputs: name, surname, date of birth, address and driving license number, which accounts to a total of 5 data elements. However, the interaction happens to be only with 1 ILF (user) and with 1 EIF (Driving Licenses external interface) and the output is still simply a "validity". As such, the inquiry is again quite simple and we consider it of low complexity.

- **Money saving option:** this inquiry requires a very simple input, that is the destination address provided by the user (which will be translated into a couple of floats representing two coordinates). Providing the output is instead quite complex, since it has to deal with several tuples of the Charging Area ILF, whose position has to be compared with the one from the user's input by going through the interaction with the Google Maps retrieval system EIF, similarly to what happened in the **Car lookup**. Furthermore, here our system is required to compare Charging Areas accordingly to several parameters (such as cars' distribution) and the output is a path consisting in the directions to reach the computed Charging Area. Considering a trade-off between the simple input and the effort to produce the output, we consider the complexity of this inquiry average.
- **View Profile:** this inquiry requires a very simple input too: just the username. As for the output, instead, it provides not only user information (all the attributes of the user entity, which were provided by the user himself during the registration), but also a history of his reservations and his rides. So, the total amount of ILF is 3 and since there are no other interactions we can consider that the complexity of this inquiry is average.

EI	Complexity	FPs
Car lookup	Average	5
Billing information checking	Low	4
Driving licenses checking	Low	4
Money saving option	Average	5
View Profile	Average	5
TOTAL		**23**

1.6 Function Points count and analysis result

Function type	FPs
Internal Logic Files	48
External Logic Files	15
External Inputs	32
External Outputs	29
External Inquiries	23
TOTAL	147

According to the QSM Function Points Languages Table[3], the multiplicator to convert the Unadjusted Function Points to Source Lines of Code for Java 2 Enterprise Edition is **46** for an *average approximation*, which is

$$SLOC = 147 \cdot 46 = 6672 \tag{1.1}$$

and **67** for a *conservative approximation*, which is

$$SLOC = 147 \cdot 67 = 9849 \tag{1.2}$$

[3] `http://www.qsm.com/resources/function-point-languages-table`

2 COCOMO II estimation

The COnstructive COst MOdel (COCOMO) is a procedural software cost estimation model. The model parameters are derived from fitting a regression formula using data from historical projects.

Measuring criteria for scale drivers and cost drivers are taken from the COCOMO II Model Definition Manual[4], where they are explained in great detail.

Cost Drivers follow the more accurate *Post-Architecture* approach.

2.1 Scale Drivers

Scale Drivers are the parameters that reflect the non-linearity of the effort with relation to the number of SLOC.

They show up at the exponent of the project size (in KSLOC) in the Effort equation (Formula 2.3).

- **Precedentedness**. If a product is similar to several previously developed projects, then the precedentedness is high. In this project, this scale driver is *Low* because we have little experience in software design and most of the notions used in this project are new to us.
- **Development flexibility**. It reflects the degree of flexibility in the development process. Low means a prescribed process needs to be followed, while High means the client only sets general goals. In this project, this scale driver is *Nominal* because we have to follow a prescribed process, but we were allowed a sufficient level of flexibility in the requirement definition and design phases.
- **Risk resolution**. It reflects the extent of the risk analysis carried out. Very Low means little analysis, Extra High means a complete and thorough risk analysis. We set it to *High*, because a rather detailed risk analysis is carried out in chapter 5 of this document.
- **Team cohesion**. It reflects how well the development team know each other and work together. Very Low means very difficult interactions, Very High means an integrated and effective team with no communication problems. We set it to *Very High*, since the cohesion among the three of us is optimal.
- **Process maturity**. It reflects the process maturity of the organization. This scale driver refers to CMM (Capability Maturity Model), a well known method for assessing the maturity of a software organization. We set it at *High*, which corresponds to *CMM Level 3 - Defined*: "It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree

[4] `http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf`

of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization."

Code	Name	Factor	Value
PREC	Precedentedness	Low	4.96
FLEX	Development flexibility	Nominal	3.04
RESL	Risk resolution	High	2.83
TEAM	Team cohesion	Very High	1.10
PMAT	Process maturity	High	3.12

The estimated scale drivers for our project, together with the formula to compute the exponent E, give a result of:

$$E = 0.91 + 0.01 \cdot \sum_{j=1}^{5} SF_j = 1.0605 \tag{2.1}$$

2.2 Cost Drivers

Cost Drivers are the parameters of the Effort Equation that reflect some characteristics of the developing process and act as multiplicators on the effort needed to build the project.

They appear as factors in the Effort equation (Formula 2.3).

In the COCOMO II Post-Architecture model, the cost drivers are 17 and they are:

Code	Name	Factor	Value	Comment
RELY	Required Software Reliability	Nominal	1.00	In case of software failure: moderate, easily recoverable losses
DATA	Data base size	Nominal	1.00	Bytes in testing database are between 10 and 100 times the estimated SLOC
CPLX	Product Complexity	Nominal	1.00	Operations performed by the software product match those of the Nominal row for this cost driver in COCOMO II Manual
RUSE	Required Reusability	High	1.07	Across program: could apply to reuse across multiple financial applications projects for the organization.
DOCU	Documentation match to life-cycle needs	Nominal	1.00	Right-sized to life-cycle needs
TIME	Execution Time Constraint	Nominal	1.00	$\leq$ 50% use of available execution time
STOR	Main Storage Constraint	Nominal	1.00	$\leq$ 50% use of available storage
PVOL	Platform Volatility	Low	0.87	For the platform(s) on which the software product is based to perform its tasks: major change every 12 months, minor change every month.
ACAP	Analyst Capability	High	0.85	Analysis, design, communication skills and efficiency of people working on requirements, high-level design and detailed design. (High: 75th percentile)
PCAP	Programmer Capability	Nominal	1.00	Programmers' ability, efficiency and communication skills. (Nominal: 55th percentile)
APEX	Application Experience	Very Low	1.22	$\leq$ 2 months
PLEX	Platform Experience	Very Low	1.19	$\leq$ 2 months
LTEX	Language and Tool Experience	Low	1.09	6 months
PCON	Personnel Continuity	Very High	0.81	Project's annual personnel turnover (Very High: 3%/year)
TOOL	Usage of Software Tools	Nominal	1.00	Basic life-cycle tools, moderately integrated
SITE	Multisite Development	High	0.93	Same city or metro area, wideband communication
SCED	Required Development Schedule	Nominal	1.00	No schedule compression or stretch-outs

Table 2.1: Cost Drivers layout and description.

The estimated cost drivers for our project, together with the formula to compute the EAF (Effort Adjustment Factor), give a result of:

$$EAF = \prod_{i=1}^{17} EM_i = 0.9432 \tag{2.2}$$

2.3 Effort

The COCOMO formula for calculating the estimated effort in person months is:

$$PM = A \cdot KSLOC^{E} \cdot EAF = \mathbf{20.8} \text{ person months} \tag{2.3}$$

Where:

- $A = 2.94$
- $KSLOC = 6.672$, as calculated in Section 1.6, Formula 1.1.
- $E = 1.0605$, as calculated in Section 2.1, Formula 2.1.
- $EAF = 0.9432$, as calculated in Section 2.2, Formula 2.2.

2.4 Duration

COCOMO II provides a simple schedule estimation capability. The initial baseline schedule equation for the COCOMO II Early Design and Post-Architecture stages is:

$$TDEV = 3.67 \cdot PM^{(0.28+0.2\cdot(E-0.91))} = \mathbf{9.4} \text{ months} \tag{2.4}$$

Where:

- $PM = 20.8$, as calculated in Section 2.3, Formula 2.3.
- $E = 1.0605$, as calculated in Section 2.1, Formula 2.1.

This would result in $\frac{PM}{TDEV} = \mathbf{2.2}$ developers needed for this project.

Since our group consists of 3 people, a reasonable development time would be (slightly approximated by excess):

$$\frac{20.8 \text{ person months}}{3 \text{ people}} \simeq \mathbf{7} \text{ months}$$

3 Tasks and schedule

The main tasks to be scheduled so that the project can be developed are:

- **RASD:** the delivery of the document containing the domain assumptions, the functional and non-functional requirement and the goals of our project.
- **DD:** the comletion of the Design Document, where the structure of the software is outlined so that the actual development can begin.
- **ITPD:** the delivery of the Integration Test Plan Document, where we devise how to integrate the components and subsystems our software is made of.
- **Project Management document:** the delivery of this document where the scheduling for each task is defined.
- **Implementation:** the actual development of the software devised in the previous documents.
- **Integration Test:** carry out the integration test planned in the ITPD in order to avoid "big bang" integration testing.
- **Presentation:** brief meeting with stakeholders where we present our project.

Dependencies

Some tasks cannot begin if other tasks are not completed:

In order to start with the **DD**, the **RASD** must be completed.

In order to start with the **Implementation**, the **DD** must be completed.

In order to start with the **ITPD**, the **DD** must be completed.

In order to start with the **Integration Test**, the **Implementation** phase must have started.

Gantt diagrams

The tasks scheduling is defined in the following Gantt diagrams. For each task, a corresponding begin and end date is defined in order to respect the dependecies we described in the previous paragraph.

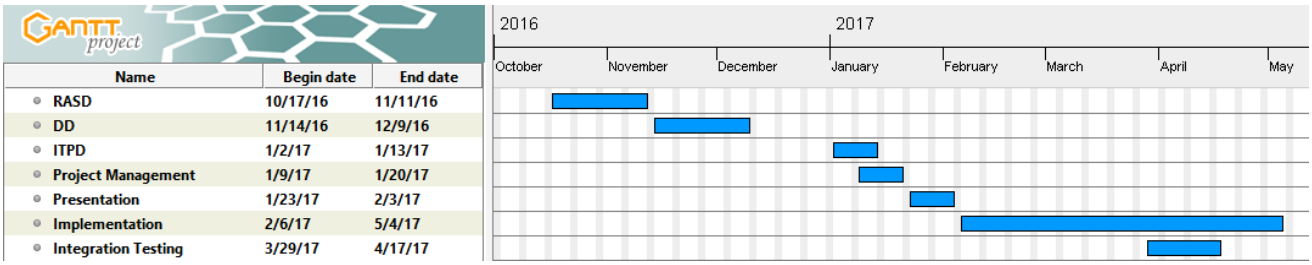

Figure 3.1: High level Gantt diagram.

For each task we provide a detailed Gantt diagram describing the sub-activities scheduling:

RASD

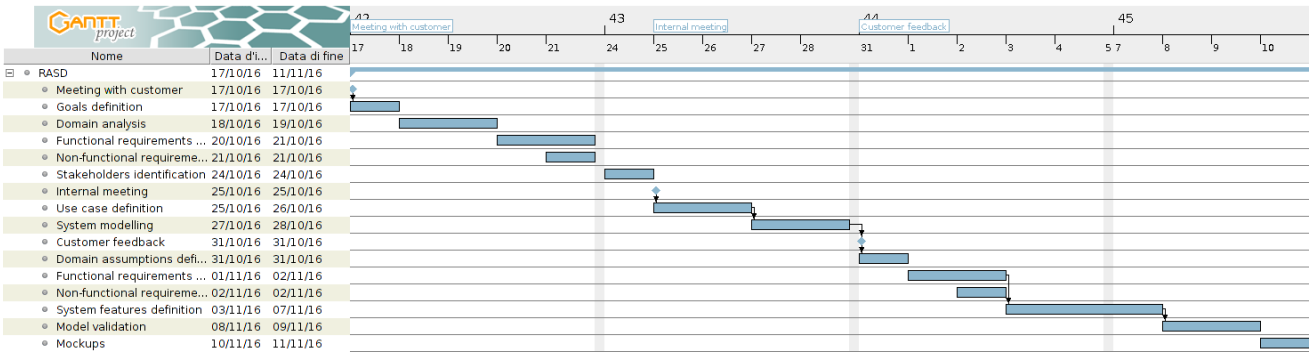

Figure 3.2: RASD detailed Gantt diagram.

DD

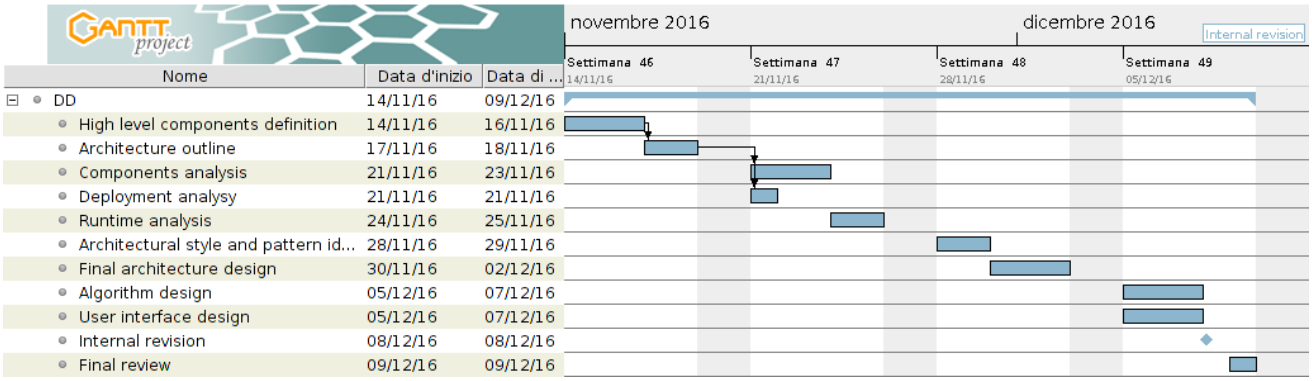

Figure 3.3: DD detailed Gantt diagram.

ITPD

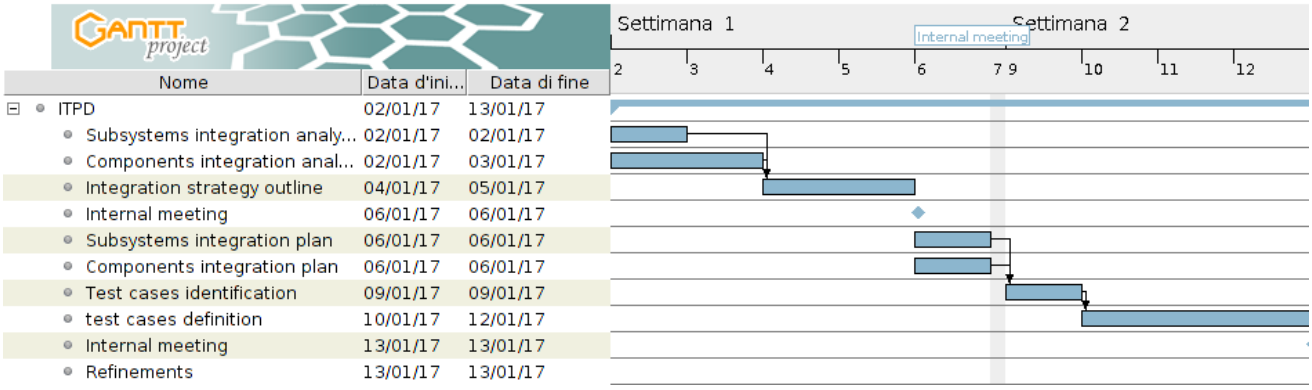

Figure 3.4: ITPD detailed Gantt diagram.

Project Management

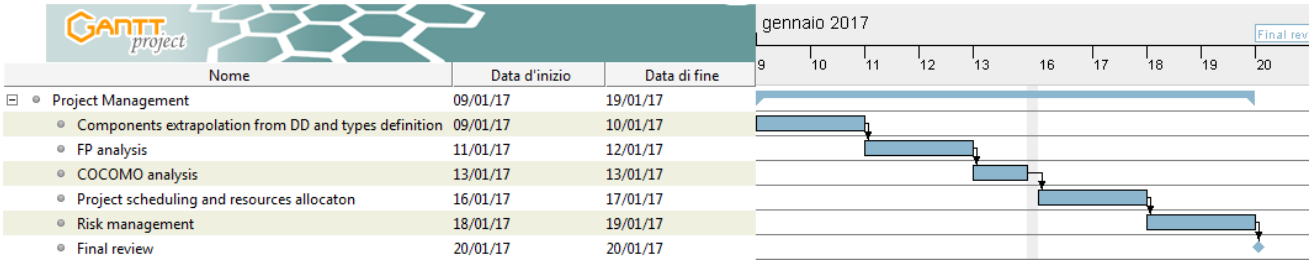

Figure 3.5: Project Management detailed Gantt diagram.

Presentation

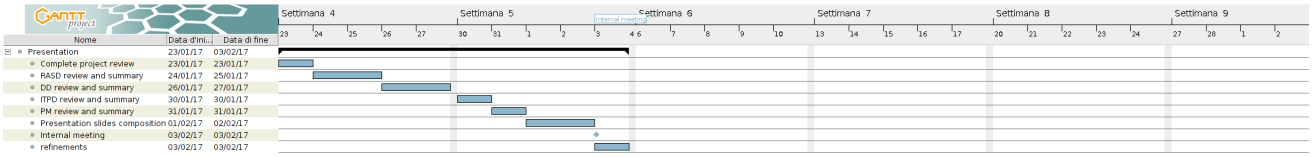

Figure 3.6: Presentation detailed Gantt diagram.

Implementation and Integration Testing

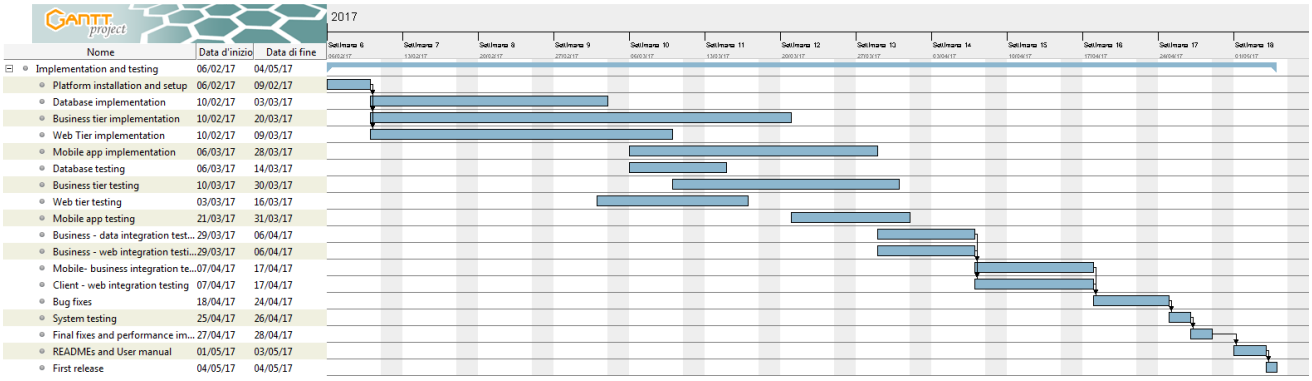

Figure 3.7: Implementation and unit testing detailed Gantt diagram.

4 Resource allocation to tasks

[intro]

Resource	**2016-10-17 to 2016-11-11**			
	1st week	2nd week	3rd week	4th week
Alessio	Spec. Req.	Overall desc.	System features	Revision Intro
Michele	Overall desc.	Intro	Spec. Req, mockups	Revision System features
Mattia	Intro	Spec. Req.	System features	Revision Overall desc.

Table 4.1: Resource allocation for RASD.

Resource	**2016-11-14 to 2016-12-09**			
	1st week	2nd week	3rd week	4th week
Alessio	Arch. design	UI design	Req. Trac.	Revision Algorithms
Michele	Algorithms	UI design	Arch. design	Revision Intro, Req. Trac.
Mattia	Intro, Req. Trac.	Arch. design	Algorithms	Revision UI design

Table 4.2: Resource allocation for DD.

Resource	**2017-01-02 to 2017-01-13**	
	1st week	2nd week
Alessio	Intro, Integration strategy	Individual steps
Michele	Intro, Integration strategy	Individual steps
Mattia	Individual steps	Revision Individual steps

Table 4.3: Resource allocation for ITPD.

Resource	**2017-01-09 to 2017-01-20**	
	1st week	2nd week
Alessio	Function Points	Tasks and schedule
Michele	COCOMO II	Resource allocation
Mattia	Function Points	Risk assessment

Table 4.4: Resource allocation for Project Management.

Resource	**2017-01-23 to 2017-02-03**	
	1st week	2nd week
Alessio		Slide
Michele		Slide
Mattia		Slide

Table 4.5: Resource allocation for Presentation.

Resource	**2017-02-06 to 2017-05-04**		
	1st month	2nd month	3rd month
Alessio	Database	Business Tier	Test Mobile
Michele	Web Tier	Mobile Client	Test Business
Mattia	Business Tier	Test Database	Test Web

Table 4.6: Resource allocation for Implementation.

Resource	**2017-03-29 to 2017-04-17**	
	1st week	2nd week
Alessio	Business - Database	Mobile - Business
Michele	Business - Database	Client - Web
Mattia	Business - Web	Mobile - Business

Table 4.7: Resource allocation for Integration Testing.

5 Risks associated with the project

5.1 Risk identification and evaluation

Risk	Detail	Probability	Effect
Lack of cooperation from team members	Team members' tasks overlap and make carrying on the project in parallel very difficult.	Low	High
Inexperience team members	Inexperienced team members require time and effort to carry on their job.	Low	Moderate
Unrealistic time and cost estimates	The actual deadline and cost for the project were slightly different the estimated ones.	Moderate	Moderate
Client loss	The client is not satisfied of the project and decides not to do business with us anymore.	Moderate	High
Misunderstanding of the requirements	It is not well understood what the software has to do in order to satisfy the goals of the project.	Low	High
Unclear or misunderstood objectives	The goal of the project are not clear and as a consequence it is also difficult to understand the requirements of the software.	Low	High
Developing the wrong software functions	The developed software does not respect the defined requirements.	Low	High
Deployment delay	The software is not ready for the deadline because one or more features are not completed.	Moderate	Low
Competitors	Other companies are building a software that is similar to the one we are developing.	Low	Moderate
Scalability and downtime issues	The software cannot bear a large number of users and this may cause the server to go down.	Moderate	Low
Legislative modification w.r.t. driving licenses	Due to legislation, driving licenses need to be validated differently.	Low	High
Bankruptcy	The company goes through financial difficulties and has no more money to carry on the project.	Low	Very High

Table 5.1: Risk identification and evaluation: probability and effects of each risk.

5.2 Adopted strategies

- **Lack of cooperation from team members**: this risk can be mitigated by adopting a schedule to decide which tasks are asssigned to each member of the group. In order to decide this, experience, skills and willingness of

the members have to be taken into account. This allows the members of the groups to work even if they are not in the same place.

- **Inexperienced team members**: inexperienced team members should be assigned less complex tasks (in terms of difficulty, but not in terms of quantity) and in case of trouble they should be helped by more experienced team members. The effort made by the latter ones will be economically compensated. In this way, experienced members are motivated to help inexperienced members who can become more skilled. Besides, the group should always be motivated and cohese in order to make everyone feel part of it.

- **Unrealistic time and cost estimates**: estimates need to be done according to other similar projects and to previous experience of the team members. Besides, the esimate is done with a pessimistic point of view, in order to take into account possible unexpected issues through which the project may go. If the first release is ready some time before the final deadline, we plan to offer technical support to the client and to ask him whether the software can be enriched with new requested features, after taking into account their feasibility in the remaining time. If the project is proceeding slowly or if it starts to cost too much, its management needs to be replanned, in particular by cutting the less necessary resources.

- **Client loss**: there are different factors that could cause the loss of the client: these may regard the software itself (it does not do what the client asked for or the interface is not appreciated, and in this case it is suggested to have frequent meetings with the client so that the software satisfies him; user acceptance tests may also help to show if it is appreciated by the stakeholders) or time constraints (the software is finished too early or too late compared to the deadline, and there are already sections where it is described what to do in this situation).

- **Misunderstanding of the software requirements**: the remedy to mitigate this risk is to keep in touch with the client: whenever something is unclear we ask further explanations to the client. It is also useful to have periodical meetings with him to show how the project is going and if it is satisfying what he asked for.

- **Unclear or misunderstood objectives**: similarly to the previous risk, the advised strategy is to ask the client for further explanations that can help the team understand and clarify doubts about the goals of the software that is under development.

- **Developing the wrong software functions**: since this is a programming issue, this risk can be easily mitigated through testing. Tests should be done starting from well defined requirements, so it is very important that the chance of misunderstanding the requirements is decreased as much

as possible. Tests have to cover the highest portion of the code that is possible so that this risk is mitigated as much as we can.

- **Deployment delay**: the adopted strategy is to deploy a first release of the software that includes its main features (for example, the ones regarding the employee may be left as the last ones to be developed as they are less critical than the ones regarding car reservation and rides). A second deadline will be discussed with the client and we are going to release the project finished in all its functionalities in a second release.

- **Competitors**: rival products need to be defeated by developing a software that is competitive in the market through the presence of innovative and appealing features.

- **Scalability and downtime issues**: a good system testing can help in understanding what is the load that the software can bear w.r.t. the number of users. Design, Architecture and code have to take into account the performance that the software needs to have in order to satisfy non-functional requirements. By increasing redundancy of the components and using load balancers the system should be up even with high loads. In case the system goes down, a well thought architecture makes repairs easy and minimizes the MTTR.

- **Legislative modification w.r.t. driving licenses**: by interfacing with an updated Driving License System we can ensure that new users' driving licenses will be verified accordingly to the latest legislations.

- **Bankruptcy**: before starting the project it is important to consider whether it is worth it or not from an economical point of view, that is whether the sale of the software will eventually be able to cover all the costs and to grant an adequate profit. A good and well-thought feasibility study can help us in the decision.

A Appendix A

A.1 Software and tools used

- LaTeX for typesetting the document.
- LyX as a document processor.
- GitHub for version control and teamwork.
- GanttProject for Gantt diagrams.

A.2 Hours of work

- Alessio Mongelluzzo: 9 hours
- Michele Ferri: 11 hours
- Mattia Maffioli: 11 hours