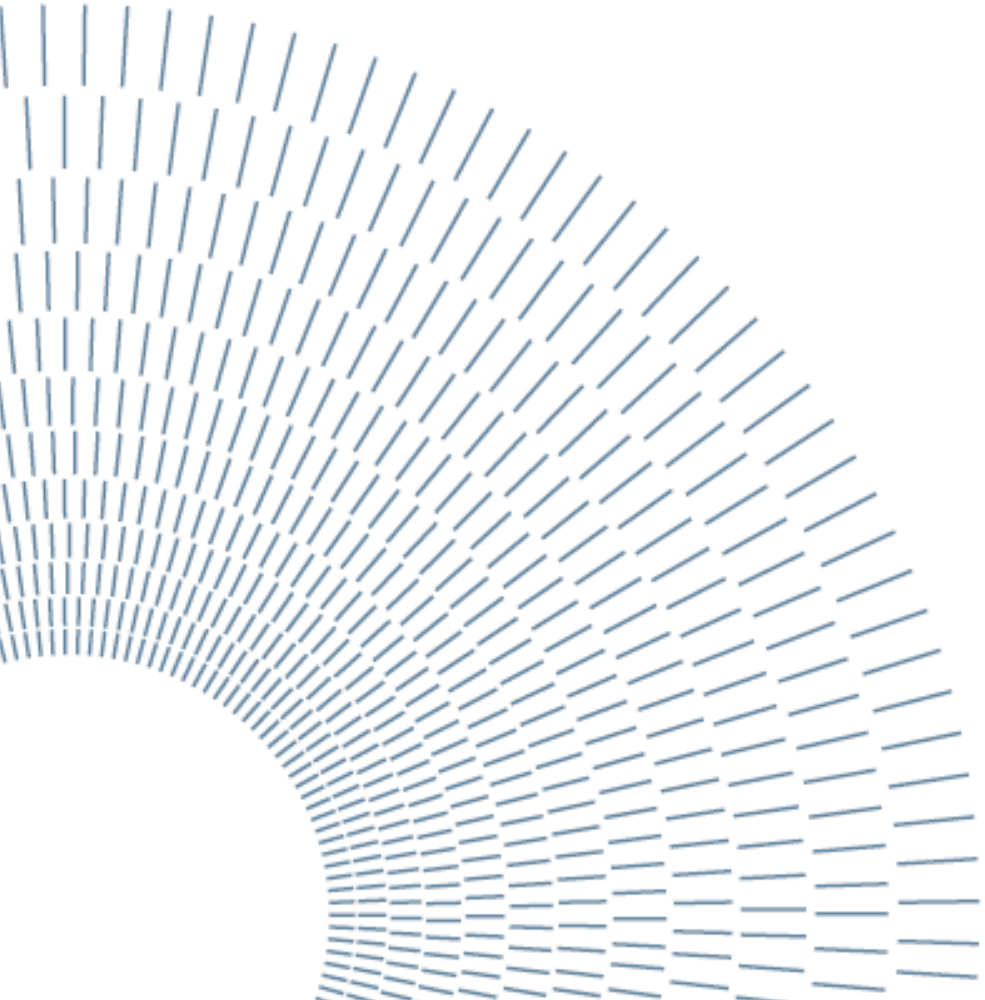
POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Software Engineering 2 2016/2017
Project Power EnJoy

Project Plan

V 1.0



Redacted by:

Leonardo Chiappalupi
CP 10453536

Ivan Bugli
CP 10453746

Table of contents

1. Introduction

1.1 Revision history

- **V1.0** / 2017.01.22 / Leonardo Chiappalupi, Ivan Bugli / Initial release

1.2 Purpose and scope

This document represents the Project Plan for Power EnJoy, and its goal is to provide an extensive analysis on the size, the costs and the manpower required to fully develop the system.

To estimate the size of the entire project, we will mainly follow the *Function Points* approach, evaluating our quantities on every function and then providing an expected size, in terms of *lines of code*.

Then, we will follow the COCOMO II approach to estimate the cost required to actually develop and build the system, providing reference tables for the different factors used.

Consequently, we will elaborate a feasible schedule for the required activities to cover the entirety of the development cycle, and then we will try to assign the various members of our team to the tasks.

In the end, we will reason about all the possible risks that the development and the future-to-be of Power EnJoy may face, providing also outlines of reasonable solutions to stem and mitigate the most severe consequences otherwise inevitable.

1.4 Definitions, acronyms, abbreviations

1.4.1 Definitions

There are no new definitions needed for the understanding of this document.

Please note that some definitions and abbreviations may carry on from the previous documents. See sections related to glossary/definitions and their subsections for further details.

1.4.2 Acronyms and abbreviations

- **FP**: Function Points.
- **ILF**: Internal logic file
- **ELF**: External logic file.
- **EI**: External Input.
- **EO**: External Output.
- **EQ**: External Inquiries.
- **DBMS**: DataBase Management System.
- **API**: Application Programming Interface.
- **ETA**: Estimated Time of Arrival.
- **GPS**: Global Positioning System.
- **UFP**: Unadjusted Function Point

1.5 Reference documents

- **Assignments document:** Assignments AA 2016-2017.pdf
- **Requirement Analysis and Specification Document:** Requirement Analysis and Specification Document V1.1.pdf
- **Design Document:** Design Document V1.2.pdf
- **Integration Testing Plan Document:** Integration Testing Plan Document V1.0.pdf
- **(*This*) Project Plan:** Project Plan V1.0.pdf
- **COCOMO II Model Manual 2000.0:** CII_modelman2000.0.pdf

2. Size, cost, effort estimation

In this part of the document we will provide several estimates of various indicators for Power EnJoy, including its expected size, cost and human effort.

This estimation will only target the specific development of the back-end logic of Power EnJoy, thus it will not examine the (likely to be out-sourced) development of user interfaces (both of mobile applications and web ones) and other presentation related features.

2.1 Size estimation

For the size estimation, we will refer to the Function Points approach, which groups the features of the system into various subsets and provides weights for every level of complexity assigned to every set of features, depending on the number of elements and record elements/file types that the system is going to use.

The tables used for this process, cured and maintained by the *International Function Point Users Group* and also available in the COCOMO II Model Manual, are reported below for convenience:

INTERNAL AND EXTERNAL LOGIC FILES

Record Elements	Data Elements *1-19*	*20-50*	*51+*
1	LOW	LOW	AVG
2-5	LOW	AVG	HIGH
6+	AVG	HIGH	HIGH

EXTERNAL OUTPUT AND EXTERNAL INQUIRY

File Types	Data Elements *1-5*	*6-19*	*20+*
0-1	LOW	LOW	AVG
2-3	LOW	AVG	HIGH
4+	AVG	HIGH	HIGH

EXTERNAL INPUT

File Types	Data Elements		
	1-4	*5-15*	*16+*
0-1	LOW	LOW	AVG
2-3	LOW	AVG	HIGH
4+	AVG	HIGH	HIGH

UFP COMPLEXITY WEIGHTS

Function Type	Complexity Weight		
	LOW	*AVG*	*HIGH*
Internal Logic Files	7	10	15
External Logic Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

2.1.1 Internal Logic Files (ILFs)

To properly run, Power EnJoy uses several ILFs – namely, the information stored in the database – organized into a precise structure. This data is managed as shown in Section 6.2: *Class Diagram* (high-level view) of the RASD and as examined in Section 2.2.2: *Persistence design* of the DD.

In the following paragraphs, we will analyze the organization of the platform ILFs, to identify their complexity and classify their weights.

The system stores information about the three kind of **users** possibly using the platform: registered users, operators and system managers. For registered users, the DB memorizes login data (email and password), name, last name, address, phone, fiscal code, driving license number, ID number, and a utility boolean to check if users are insolvent or not. For the other two kinds of users, we just retain a username, a password, and an id to match them to the various operations. Speaking from a relational database point of view, for users management we will need a total of three tables, two of which are very simple.

Cars are stored into the system by saving their plate, their battery level, and two convenience booleans to determine if, at the last update, the car was plugged or not and if the car is enabled or not. The car table also has a column for the status of the car (reservable/charging/reserved/out-of-order/etc.). A second table matched every car to its last updated position, in terms of coordinates. So, car management ILFs are slightly more complex than user management ones.

As for **parking areas**, they are stored in a single table, which contains their name, a bool to determine if it's a parking area with plugs or not, and nothing else. To store the bounds though, we will use another table that matches all the bounds (coordinates) to every parking area, thus memorizing the "polygon" of the actual area: this two-level structure is quite complex.

Moving to the service itself, we have a table to manage **reservations**: it contains an id, the username/email of the user that made it, the date and the time when it has been made, and a reference to the car actually reserved. A convenience reference in the users table matches every user to its active reservation, if any. So, the structure for reservations is not particularly complex.

Then, we have **rides**: they are stored in terms of car used, user who started it, total cost, information about the payment correctness, the date, time of start and time of end. Another table matches every ride to the conditions of the car when it was parked: this includes a reference to the parking area, the battery left on the car, if the car was plugged or not, and if not, if the car was further than 3Km from the closest plug parking area. A third table matches every ride to every sharing user possibly sharing that ride. Extras in the price (surcharges and discounts) are stored in a dedicated table to allow future developments: this data is made of the name of the extra, the type (bonus or minus), the percentage, and a description of the policies. Another table matches every ride to every extra eventually applied when paying. In conclusion, the ILFs for rides are somewhat complex.

The DB also stores information about **assistance requests**, using an id, a reference to the user which made the request, the date and the time, and a reference to the related ride, in addition to two bools to determine if the request has been forwarded to operators or is still for system managers, and if it has been solved or not. Another table matches every operator to the assistance request he is possibly taking charge of or that did in the past.

Referring to the tables listed at the beginning of the main section for Internal Logic Files, the following list shows the complexity and the FPs assigned for every kind of data the system manages:

Internal Logic File	Complexity assigned	Corresponding FPs
Users data	LOW	7
Car data	AVG	10
Parking areas	AVG	10
Reservations	LOW	7
Rides	HIGH	15
Assistance requests	LOW	7
Total Function Points for ILFs		**56**

2.1.2 External Logic Files (ELFs)

Power EnJoy relies upon some external data to work. As already stressed in previous documents, our service will use an external *Payment Handler* to manage transactions, and it will make also use of Google Maps APIs to convert between coordinates and addresses – and vice versa (this interaction has been always not explicitly represented in diagrams but is carrying from the *Requirement Analysis and Specification Document*). Maps APIs will also be used for more complex operations on the client application side – like associating marks of cars and parking areas with points on a map, estimating the distance between the user and reservable cars, etc.

The first set of ELFs – the one related to payments – is indeed fairly simple: it will consist of JSONs containing payments details, sent back and forth using RESTful operations. The problem is that we will also have to manage the security of the connection, thus providing authorization tokens and managing sessions.

As for Google Maps ELFs, the amount of data required for accessing and using the service is quite large too: not only we will need to manage authorization tokens to use the service, but we will also need to process a lot of data from maps to retrieve the actual information we need.

Consequently, we associate the following complexities and relative Function Points to the features described above, always using reference tables listed at the beginning:

External Logic File	Complexity assigned	Corresponding FPs
Payments	AVG	7
Coordinates conversions	HIGH	10
Other maps information	HIGH	10
Total Function Points for ELFs		**27**

2.1.3 External Inputs (EIs)

Power EnJoy offers a wide variety of features, and they are performed by responding to external inputs provided by the various kinds of users that will use the service. In this sub-section, we will analyze how these various inputs trigger operations of different levels of complexity, and we will assign FPs depending on this criterion.

Starting from users, they can:

- **Login:** very basic operation, involving two components. The complexity is LOW.
- **Register:** this operation has an average complexity, considering the amount of data exchanged and the various operations to verify its consistency and generate the password. Since only two components are involved, the complexity is AVG.
- **Update personal information:** basic operation, consisting in receiving and storing in the DB. Thus, the complexity is LOW.
- **Reserve a car:** this input triggers complex operations, such as managing 1-hour timers and changing the reserved car's state. This operation may involve up to 3 components, if we consider the case of paying a fee: the assigned complexity is HIGH.
- **Delete a reservation:** this is not a complex operation, and only two components are interested. Complexity is LOW.
- **Browse rides/reservations history:** from a logic point of view, these are very easy operations, thus their complexity is LOW.
- **Unlock a car (start a ride):** this is a very complex input to manage, since it requires the validation of positions, the transition from a reservation to a ride, the management of the ignition trigger from the car, etc. It also involves a lot of components; thus, the complexity is HIGH.
- **Lock a car (end a ride):** this is also a complex flow of operations, that involves a lot of components, for instance checking the termination conditions and the status of the car. Assigned complexity is therefore HIGH.
- **Share another user's ride:** this is an easy request to manage, since it only consists in associating a username with a ride. Complexity: LOW.
- **Retrieve cars in range (address/position):** quite complex operation, involving the interaction with external components and filtering a high number of data. The number of components involved is nonetheless low: the complexity given will be AVG.
- **Create an assistance request:** basic operation and small input to process. The complexity in this case is indeed LOW.

System manager may:

- **Retrieve the fleet of cars/parking areas:** easy request, from a logical point of view, and small input. But data processed may be high, and there may be the need of using external map data: complexity is AVG.
- **Update a parking area:** parking areas' bounds may be quite a lot, and therefore there could be the necessity of validating a lot of data. Even if involved components are a few, the update processing has, overall, a HIGH complexity.
- **Remove a parking area:** easy removal operation, by name. Complexity is then LOW.
- **Add a parking area:** bounds may be complicate to receive and process, but this is only a storing operation: complexity is overall AVG.
- **Add car to the fleet:** cars data is updated by cars themselves, so a system manager may only use a few details (such as the plate) to add a new car. The complexity of this operation is LOW.
- **Remove car from the fleet:** very basic removal operation, small input: complexity is LOW.
- **Retrieve list of assistance requests:** a basic query that does not involve a lot of data on the input side: complexity is LOW.
- **Forward a request to operators:** this operation uses a small input but activates in cascade several components, and implies some non-trivial database operations, too: complexity is AVG.
- **Mark an assistance request as solved:** easy triggered operation, with complexity that is then LOW.

Operators, instead, can:

- **Get a list of out-of-order and unavailable cars:** basic input and straightforward query. Complexity is LOW.
- **Retrieve assistance requests (assigned to operators):** as above, very trivial operation. LOW complexity then.
- **Take charge of a car/assistance request:** this involves only the components dedicated to the operators and implies to just flag a car/assistance request as assigned to someone. Complexity is LOW.
- **Mark an issue as solved (on a car):** this triggers several operations to move the car from the out-of-order state to a new state, depending on its conditions. A number of components is required here, thus the operation has an AVG complexity.

- **Mark a request as solved:** this only implies flagging an assistance request as solved (after the intervention of an operator on-site): thus, the complexity is LOW.

Finally, cars can also spontaneously provide data to the system in several cases. They may:

- **Report their status:** the input data contains all the information about the car and its status, that must be stored in the DB: complexity is AVG.
- **Ask for the status of a ride:** to show information in the on-board computer, cars may query the system for information. The operation, however, is simple: its complexity is then LOW.
- **Ask to be re-enabled after a charge:** cars that had low battery and have been charged over 80% must signal it to be flagged as available again. This operation involves updating car data in the database and enabling the car: the complexity is AVG.
- **Report ignited:** cars report when they are ignited, and this triggers a variety of operations during the ride start phase. Thus, the complexity is AVG.
- **Report parked:** cars report when they are turned off (parked), and in which parking area. This has no complex input or consequences, thus the complexity assigned is LOW.

Now that all the external inputs have been examined, we can provide a summary table with the total count for FPs:

Table on the next page

External input(s)	Complexity assigned	Corresponding FPs
Login	LOW	3
Register	AVG	4
Update information	LOW	3
Reserve a car	HIGH	6
Delete a reservation	LOW	3
Browse reservations or rides	LOW	3
Unlock a car	HIGH	6
Lock a car	HIGH	6
Share a ride	LOW	3
Retrieve car in range	AVG	4
Create assistance req.	LOW	3
Retrieve fleet/park. areas	AVG	4
Update parking area	HIGH	6
Remove Parking area	LOW	3
Add parking area	AVG	4
Add new car	LOW	3
Remove car from fleet	LOW	3
Retrieve assistance requests	LOW	3
Forward request to operators	AVG	4
Mark request solved	LOW	3
Get unavailable cars	LOW	3
Retrieve requests for operators	LOW	3
Take car or assistance request	LOW	3
Mark issue solved	AVG	4
Mark request solved	LOW	3
Report car status	AVG	4
Query ride status	LOW	3
Re-enable after charge	AVG	4
Report ignited	AVG	4
Report parked	LOW	3
Total Function Points for EIs		**111**

2.1.4 External Outputs (EOs)

Normally, Power EnJoy does not need to communicate with the users or the car spontaneously, i.e. outside the normal flow of request and responses already examined. There are some situations though in which this may happen, specifically:

- **Password after registration**: after users have registered, we need to send their password via e-mail. This is obviously a very simple task, with a LOW complexity.
- **Car ride data updates**: every minute, the *RideController* component sends the car an update on the time of the current ride, to allow the car to compute the price and show it on the on-board screen. Since this feature involves some components, its complexity is AVG.
- **Car locking and unlocking**: these are commands that the system issues to the car, independently from the car itself, so we think they belong here: since they involve different components and the car applet, we think their complexity is AVG.

The following table summarizes the new Function Points just introduced:

External Output	Complexity assigned	Corresponding FPs
Password after registration	LOW	4
Car ride updates	AVG	5
Car locking/unlocking	AVG	5 (x2)
Total Function Points for EOs		**19**

2.1.5 External Inquiries (EQs)

External Inquiries are the requests, performed by clients, that involve the retrieval of a set of information.

Power EnJoy offer some of these operations for all the clients possibly using or involved in the service, including the car:

Registered users can:

- **Retrieve a list of their past rides/reservations:** easy operations, limited to returning a limited set of queries results: complexity is LOW.
- **Retrieve the positions of nearby cars/parks OR cars/parks by address:** quite complex operations, that also involves the external mapping service provider: complexity is HIGH.

- **Get data about the current reservation, if any:** this is a simple operation, and the chosen complexity is LOW.

System managers may:

- **Get a list of the cars in the fleet:** easy operation, limited to a few components and the database: LOW complexity.
- **Get a map with parking areas overlaid:** this is a complex operation, involving maps and the processing of a huge amount of data for the bounds of the areas. The chosen complexity, hence, is HIGH.
- **Get a list of assistance requests:** very easy operation, limited to a query with some filters. Its complexity is LOW.

Operators, on their side, can:

- **Get a list of unavailable/out-of-order cars:** fast operation, especially given the easy data structure for cars: complexity is LOW.
- **Get a list of assistance requests (for operators):** easy process, which won't likely even regard a big amount of data. We assign a LOW complexity.

Finally, cars can retrieve the **ride status of the one they are being used for**, which is a fairly simple operation and that will have a LOW complexity assigned.

The following table recaps the EQs complexities and Function Points, computing the total:

External Inquiry	Complexity assigned	Corresponding FPs
Past rides/reservations	LOW	3 (x2)
Nearby cars/parks	HIGH	6
Address cars/parks	HIGH	6
Current reservation	LOW	3
Cars in the fleet	LOW	3
Parking areas	HIGH	6
Assistance requests	LOW	3
Unavailable cars	LOW	3
Assistance request operators	LOW	3
Total Function Points for EQs		**39**

2.1.6 Final size estimation

Now that the various kinds of Function Points have been assigned, we can compute the total FPs scale and then use it to get some bounds in terms of lines of code.

Function points have been assigned as summarized in the following table:

Function Type	Function Points
Internal Logic Files	56
External Logic Files	27
External Inputs	111
External Outputs	19
External Inquiries	39
Total Function Points	**252**

Using the QSM Function Points languages table (see references in Section 6: *Appendices*), we can now calculate an estimate of the size of the project, in terms of lines of code, using the J2EE parameter (since we will use Java Enterprise Edition to build the platform). Note that these numbers do not include the code that will be necessary to develop the mobile applications and the web applications, that are not part of the core back-end of the system we are evaluating.

The factors that we are going to use are:

Estimation (factor)	Min	Avg	Max
Conversion rate	15	46	67

And the final calculations are:

```
SLOC_MIN = 15 * 252 =  3.780
SLOC_AVG = 46 * 252 = 11.592
SLOC_MAX = 67 * 252 = 16.884
```

This means that we have a lower bound of 3 780 lines of code, an average estimation of 11 592 and an upper bound of 16 884 lines.

2.2 Cost and effort estimation

Following closely the COCOMO II approach, we will now estimate both the effort and the time expected for the development of the project, going through a series of *Scale drivers* and *Cost drivers* and associating a factor for every one of them to Power EnJoy. Then, we will use the computed totals to make the estimations.

Every driver will have the official table quoted on the document, followed by a brief explanation and the factor chosen consequently.

2.2.1 Scale drivers

PREC – PRECEDENTEDNESS

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
PREC	THOROUGHLY UNPRECEDENTED	LARGELY UNPRECEDENTED	SOMEWHAT UNPRECEDENTED	GENERALLY FAMILIAR	LARGELY FAMILIAR	THOROUGHLY FAMILIAR
SF_j	6.20	**4.96**	3.72	2.48	1.24	0.00

Previous experience of the team with projects of this kind and scale. Our time is LARGELY UNPRECEDENTED with them, so the factor will be LOW (4.96).

FLEX – DEVELOPMENT FLEXIBILITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
FLEX	RIGOROUS	OCCASIONAL RELAXATION	SOME RELAXATION	GENERAL CONFORMITY	SOME CONFORMITY	GENERAL GOALS
SF_j	5.07	4.05	**3.04**	2.03	1.01	0.00

The degree of freedom for development in relation to requirements and specifications. For this project, we had quite strict requirements on functionalities but almost none on technologies, so we will pick SOME RELAXATION and this factor will be set to NOMINAL (3.04).

RESL – RISK RESOLUTION

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
RESL	LITTLE (20%)	SOME (40%)	OFTEN (60%)	GENERALLY (75%)	MOSTLY (90%)	FULL (100%)
SF_j	7.07	5.65	4.24	2.83	**1.41**	0.00

Represent the level of analysis and preparation that our team is likely to have in response to risks realization. The analysis that we performed on risks in Section 5 is extensive, so we have MOSTLY (90%) risk resolution and the factor will be set to VERY HIGH (1.41).

TEAM – TEAM COHESION

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
RESL	VERY DIFFICULT INTERACTIONS	SOME DIFFICULT INTERACTIONS	BASICALLY COOPERATIVE INTERACTIONS	LARGELY COOPERATIVE INTERACTIONS	HIGHLY COOPERATIVE INTERACTIONS	SEAMLESS INTERACTIONS
SF_j	5.48	4.38	3.29	2.19	**1.10**	0.00

As the name suggests, this is indicator of the cohesion of the development team, and its tendency to cooperate in a teamwork. We are quite confident on our skills in this sense, but not to overestimate the factor we will choose HIGHLY COOPERATIVE, with a factor set to VERY HIGH (1.10).

PMAT – PROCESS MATURITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
PMAT	LEVEL 1 LOWER	LEVEL 1 UPPER	LEVEL 2	LEVEL 3	LEVEL 4	LEVEL 5
SF_j	7.80	6.24	**4.68**	3.12	1.56	0.00

The maturity of the team in the project and development process. Since we have no previous experience on this kind of high-scale projects, and we are fulfilling goals for the first time but successfully, we estimate our maturity to be LEVEL 2, thus resulting in a NOMINAL (4.68) factor.

Our final evaluation on scale factors is then summarized as follows:

Scale Driver	Factor	Value
PREC – PRECEDENTEDNESS	LOW	4.96
FLEX – DEVELOPMENT FLEXIBILITY	NOMINAL	3.04
RESL – RISK RESOLUTION	VERY HIGH	1.41
TEAM – TEAM COHESION	VERY HIGH	1.10
PMAT – PROCESS MATURITY	NOMINAL	4.68
Scale factors total		**15.19**

2.2.2 Cost drivers

RELY – REQUIRED SOFTWARE RELIABILITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
RELY	SLIGHTLY INCONVENIENCE	EASILY RECOVERABLE LOSSES	MODERATE RECOVERABLE LOSSES	HIGH FINANCIAL LOSS	RISK TO HUMAN LIFE	-
EFFORT MUL.	0.82	0.92	1.00	**1.10**	1.26	-

The platform we are designing and developing has a lot of competitors: hence, any failure or malfunctioning could have a huge impact on users' confidence, resulting in high economic losses. The reliability could lead to HIGH FINANCIAL LOSS, so the rating level will be HIGH (1.10).

DATA – DATABASE DRIVERS

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
DATA	-	$\frac{D}{P}<10$	$10\leq\frac{D}{P}\leq10$	$100\leq\frac{D}{P}\leq1000$	$\frac{D}{P}>1000$	-
EFFORT MUL.	-	0.90	1.00	**1.14**	1.28	-

We expect our database, comparing it to similar ones, to be not over 5Gb in size. This is only an estimation, but reasonable if compared to the actual DB structure. Since this database is used by an application which is – by average – 11.592 SLOC, our D/P ratio (database size in bytes/SLOC) results in about 431, that assigns a HIGH (1.14) rating level.

CPLX – PRODUCT COMPLEXITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
EFFORT MUL.	0.73	0.87	1.00	1.17	**1.34**	1.74

Basing on the rating scale levels, we state that the product complexity for Power EnJoy is VERY HIGH (1.34).

RUSE – REQUIRED REUSABILITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
RUSE	-	NONE	ACROSS PROJECT	ACROSS PROGRAM	ACROSS PRODUCT LINE	ACROSS MULTIPLE PRODUCT LINES
EFFORT MUL.	-	0.95	**1.00**	1.07	1.15	1.24

We do not have reusability constraints or requirements, except for the boundaries of the project itself, that must be somehow modular and allow reusability of

components. So, the required usability is ACROSS PROJECT and the rating level resulting is NOMINAL (1.00).

DOCU – DOCUMENTATION MATCH TO LIFE-CYCLE NEEDS

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
DOCU	MANY LIFE-CYCLE NEEDS UNCOVERED	SOME LIFE-CYCLE NEEDS UNCOVERED	RIGHT-SIZED TO LIFE-CYCLE NEEDS	EXCESSIVE FOR LIFE-CYCLE NEEDS	VERY EXCESSIVE FOR LIFE-CYCLE NEEDS	-
EFFORT MUL.	0.81	**0.91**	1.00	1.11	1.23	-

The ratio between the documentation produced or planned with the actual needs, during the life-cycle, of the system itself. We did not present some necessary complementary documents (e.g. the manual), so we will set this to SOME LIFE-CYCLE NEEDS UNCOVERED, that corresponds to a LOW (0.91) rating level.

TIME – EXECUTION TIME CONSTRAINT

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
TIME	-	-	≤ 50% USE OF AVAILABLE EXECUTION TIME	70% USE OF AVAILABLE EXECUTION TIME	85% USE OF AVAILABLE EXECUTION TIME	95% USE OF AVAILABLE EXECUTION TIME
EFFORT MUL.	-	-	1.00	1.11	**1.29**	1.63

Power EnJoy, when a reasonable audience of users will be reached, will likely have its servers very stressed – especially during some particular days and hours. We expect our servers to make around 85% USE OF AVAILABLE EXECUTION TIME, so the rating level is VERY HIGH (1.29).

STOR – STORAGE CONSTRAINT

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
STOR	-	-	≤ 50% USE OF AVAILABLE STORAGE	70% USE OF AVAILABLE STORAGE	85% USE OF AVAILABLE STORAGE	95% USE OF AVAILABLE STORAGE
EFFORT MUL.	-	-	1.00	**1.05**	1.17	1.46

As described above, we do not expect our database to be over around 5Gb of persistent data. This parameter measures the usage of the overall storage space available; since we are planning to deploy Power EnJoy on Amazon AWS, whose available space follows a pay-as-you-go model, we do not expect to reserve much more space than that, so we will set around 70% USE OF AVAILABLE STORAGE with a HIGH (1.05) resulting rating level.

PVOL – PLATFORM VOLATILITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
PVOL	-	MAJOR CHANGE: 12 MO. MINOR CHANGE: 1 MO.	MAJOR CHANGE: 6 MO. MINOR CHANGE: 2 WK.	MAJOR CHANGE: 2 MO. MINOR CHANGE: 1 WK.	MAJOR CHANGE: 2 WK. MINOR CHANGE: 2 DAYS	-
EFFORT MUL.	-	**0.87**	1.00	1.15	1.30	-

We do not expect to make major changes on the backend very often, neither on client apps since mobile OS SDKs have reached a good level of backward compatibility thus not requiring lots of changes at every OS release. Moreover, we designed the system to be very scalable so changes in the size of the system won't require major breakdowns. Consequently, we expect MAJOR CHANGES EVERY 12 MO., MINOR CHANGES EVERY 1 MO., that corresponds to a LOW (0.87) rating level.

ACAP – ANALYST CAPABILITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
ACAP	15TH PERCENTILE	35TH PERCENTILE	55TH PERCENTILE	75TH PERCENTILE	90TH PERCENTILE	-
EFFORT MUL.	1.42	1.19	**1.00**	0.85	0.71	-

Even if our team main occupation is not this kind of analysis, we are quite confident that the one of this project has been carried over carefully; however, to avoid the risk of over-estimating our abilities, we set the descriptor to 55TH PERCENTILE, that is a NOMINAL (1.00) rating level.

PCAP – PROGRAMMER CAPABILITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
PCAP	15TH PERCENTILE	35TH PERCENTILE	55TH PERCENTILE	75TH PERCENTILE	90TH PERCENTILE	-
EFFORT MUL.	1.34	1.15	1.00	**0.88**	0.76	-

Since at the moment of this document's writing, our team did not yet start implementing the project into code, we can make just an estimation here; we are pretty confident about our programming skills, however we will stay conservative as already did before and we will set this parameter to 75TH PERCENTILE, that corresponds to a HIGH (0.88) rating level.

APEX – APPLICATION EXPERIENCE

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
APEX	≤ 2 MONTHS	6 MONTHS	1 YEAR	3 YEARS	6 YEARS	-
EFFORT MUL.	**1.22**	1.10	1.00	0.88	0.81	-

We have about one year and a half of experience in developing large Java applications, but we have only a few months of experience with Java Enterprise Edition. Combined with the fact that we never developed an application with this features, and to stay conservative, we will choose the ≤ 2 MONTHS descriptor, resulting in a VERY LOW (1.22) rating level.

PLEX – PLATFORM EXPERIENCE

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
APEX	≤ 2 MONTHS	6 MONTHS	1 YEAR	3 YEARS	6 YEARS	-
EFFORT MUL.	1.19	1.09	**1.00**	0.91	0.85	-

We have to consider three different factors that sum up to make our experience with the *platform*: our experience with the various mobile SDKs, that is average, our experience with the Amazon AWS setup, that is quite high, and our experience with applications that use a client-side architecture and a database, which is high. But since we have a moderate experience with Java EE, we will set our platform experience to 1 YEAR, that is NOMINAL (1.00) rating level.

LTEX – LANGUAGE AND TOOL EXPERIENCE

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
APEX	≤ 2 MONTHS	6 MONTHS	1 YEAR	3 YEARS	6 YEARS	-
EFFORT MUL.	1.20	**1.09**	1.00	0.91	0.84	-

Our team as a very low experience with the Java EE framework but a quite high one with Java itself; nonetheless the experience with testing tools and some other ones is very low, so we estimate to have an equivalent 6 MONTHS experience for this descriptor that corresponds to a LOW (1.09) rating level.

PCON – PERSONNEL CONTINUITY

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
PCON	48% / YEAR	24% / YEAR	12% / YEAR	6% / YEAR	3% / YEAR	-
EFFORT MUL.	1.29	**1.12**	1.00	0.90	0.81	-

For the continuity of the personnel developing and maintaining the project, our team will spend a limited period of time and then will likely be replaced by other members or developers. We expect the continuity to be broken quite often even after this first phase, so we estimate that a suitable descriptor could be 24%/YEAR, thus resulting in a LOW (1.12) rating level.

TOOL – USAGE OF SOFTWARE TOOLS

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
TOOL	EDIT, CODE, DEBUG	SIMPLE, FRONTEND, BACKEND CASE, LITTLE INTEGRATION	BASIC LIFE-CYCLE TOOLS, MODERATELY INTEGRATED	STRONG, MATURE LIFE-CYCLE TOOLS, MODERATELY INTEGRATED	STRONG, MATURE, PROACTIVE LIFE-CYCLE TOOLS, WELL INTEGRATED WITH PROCESSES, METHODS, REUSE	-
EFFORT MUL.	1.17	1.09	1.00	**0.90**	0.78	-

We think that our application, as it is described in the documents so far, will provide STRONG, MATURE LIFE-CYCLE TOOLS, MODERATELY INTEGRATED, so we will set a HIGH (0.90) rating level.

SITE – MULTISITE DEVELOPMENT

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
SITE COLLOCATION DESCRIPTORS	INTERNATIONAL	MULTI-CITY AND MULTI-COMPANY	MULTI-CITY OR MULTI-COMPANY	SAME CITY OR METRO AREA	SAME BUILDING OR COMPLEX	FULLY COLLOCATED
SITE COMMUNICATIONS DESCRIPTORS	SOME PHONE, MAIL	INDIVIDUAL PHONE, FAX	NARROW BAND EMAIL	WIDEBAND ELECTRONIC COMMUNICATION	WIDEBAND ELECTRONIC COMMUNICATION, OCCASIONAL VIDEO CONFERENCE	INTERACTIVE MULTIMEDIA
EFFORT MUL.	1.22	1.09	1.00	**0.93**	0.86	0.80

Our team is made of two people that live in different cities; however, we had the chance of using WIDEBAND ELECTRONIC COMMUNICATION (social networks, skype, etc.) and in addition, the two cities are in the same METRO AREA: these descriptors fit the HIGH (0.93) rating level.

SCED – REQUIRED DEVELOPMENT SCHEDULE

Factor	VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH
SCED	75% OF NOMINAL	85% OF NOMINAL	100% OF NOMINAL	130% OF NOMINAL	160% OF NOMINAL	-
EFFORT MUL.	1.43	1.14	1.00	**1.00**	1.00	-

For the Power EnJoy design, analysis and development, our team has always been quite regular into the schedules, even if for the first two documents we think that the time required was higher than nominal. A plausible estimation at this point is the 130% OF NOMINAL descriptor, which results in a HIGH (1.00) rating level.

Our final evaluation on cost factors is then summarized as follows:

Scale Driver	Factor	Value
RELY – REQUIRED SOFTWARE RELIABILITY	HIGH	1.10
DATA – DATABASE DRIVERS	HIGH	1.14
CPLX – PRODUCT COMPLEXITY	VERY HIGH	1.34
RUSE – REQUIRED REUSABILITY	NOMINAL	1.00
DOCU – DOCUMENTATION MATCH TO LIFE-CYCLE NEEDS	LOW	0.91
TIME – EXECUTION TIME CONSTRAINT	VERY HIGH	1.29
STOR – STORAGE CONSTRAINT	HIGH	1.05
PVOL – PLATFORM VOLATILITY	LOW	0.87
ACAP – ANALYST CAPABILITY	NOMINAL	1.00
PCAP – PROGRAMMER CAPABILITY	HIGH	0.88
APEX – APPLICATION EXPERIENCE	VERY LOW	1.22
LTEX – LANGUAGE AND TOOL EXPERIENCE	LOW	1.09
PCON – PERSONNEL CONTINUITY	LOW	1.12
TOOL – USAGE OF SOFTWARE TOOLS	HIGH	0.90
SITE – MULTISITE DEVELOPMENT	HIGH	0.93
SCED – REQUIRED DEVELOPMENT SCHEDULE	HIGH	1.00
Cost drivers total (product)		**1.9767**

2.2.3 Effort

Using some equations, we are now able to calculate the estimated effort in Person-Months (PM). The equation is:

$$EFFORT = A * EAF * KSLOC^{E}$$

The meaning or the value of the parameters is the following:

A = **2.94** (COCOMO II)

EAF = Cost drivers total product (**1.9767**)

E = Exponent of scale drivers.
Derived as: B + 0.01 * $\sum(i)$ $SF[i]$ = B + 0.01 * 15.19 = **1.0619**
(B is 0.91 per COCOMO II)

Since we provided three different estimations for the SLOC value (lower-bound, average, upper-bound), we will compute the effort value for the three cases:

$$EFFORT\text{-}Lower = 2.94 * 1.9767 * 3.780^{1.0619} = 23.8521\ PM \approx \mathbf{24\ PM}$$

$$EFFORT\text{-}Avg = 2.94 * 1.9767 * 11.592^{1.0619} = 78.4003\ PM \approx \mathbf{78\ PM}$$

$$EFFORT\text{-}Upper = 2.94 * 1.9767 * 16.884^{1.0619} = 116.8810\ PM \approx \mathbf{117\ PM}$$

2.2.4 Time estimation

To compute an estimation of the final duration, we will use the formula:

$$DURATION = 3.67 * EFFORT^{F}$$

$$F = 0.28 + 0.2 * (E - B) = 0.28 + 0.2 * (1.0619 - 0.91) = \mathbf{0.31038}$$

Parameters E and B are to be computed as in the previous subsection. For the lower-bound, average and upper-bound values of the *Effort*, we finally find:

$$DURATION\text{-}Lower = 3.67 * 23.8521^{0.31038} = \mathbf{9.82\ Months}$$

$$DURATION\text{-}Avg = 3.67 * 78.4003^{0.31038} = \mathbf{14.21\ Months}$$

$$DURATION\text{-}Upper = 3.67 * 116.8810^{0.31038} = \mathbf{16.09\ Months}$$

All these values seem reasonable durations for a project of this size and complexity.

3. Schedule

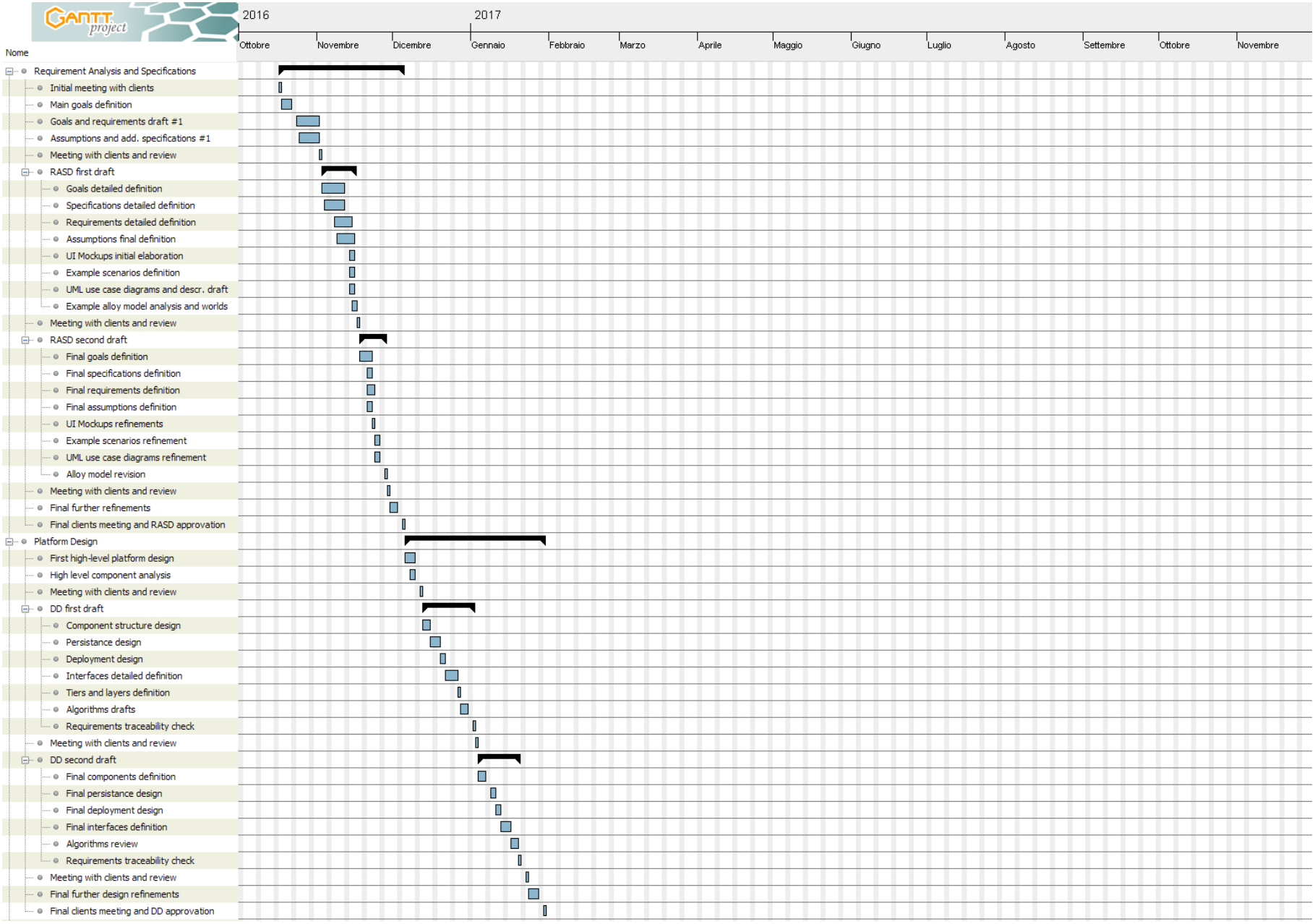
This section will introduce a general schedule for the entire development cycle of this project, from the first meeting with our clients to the final public release. To be more precise with the activities planning, we also included meetings, deployment phases, documents reviews, and any other activity that is relevant in the scheduling of the project.

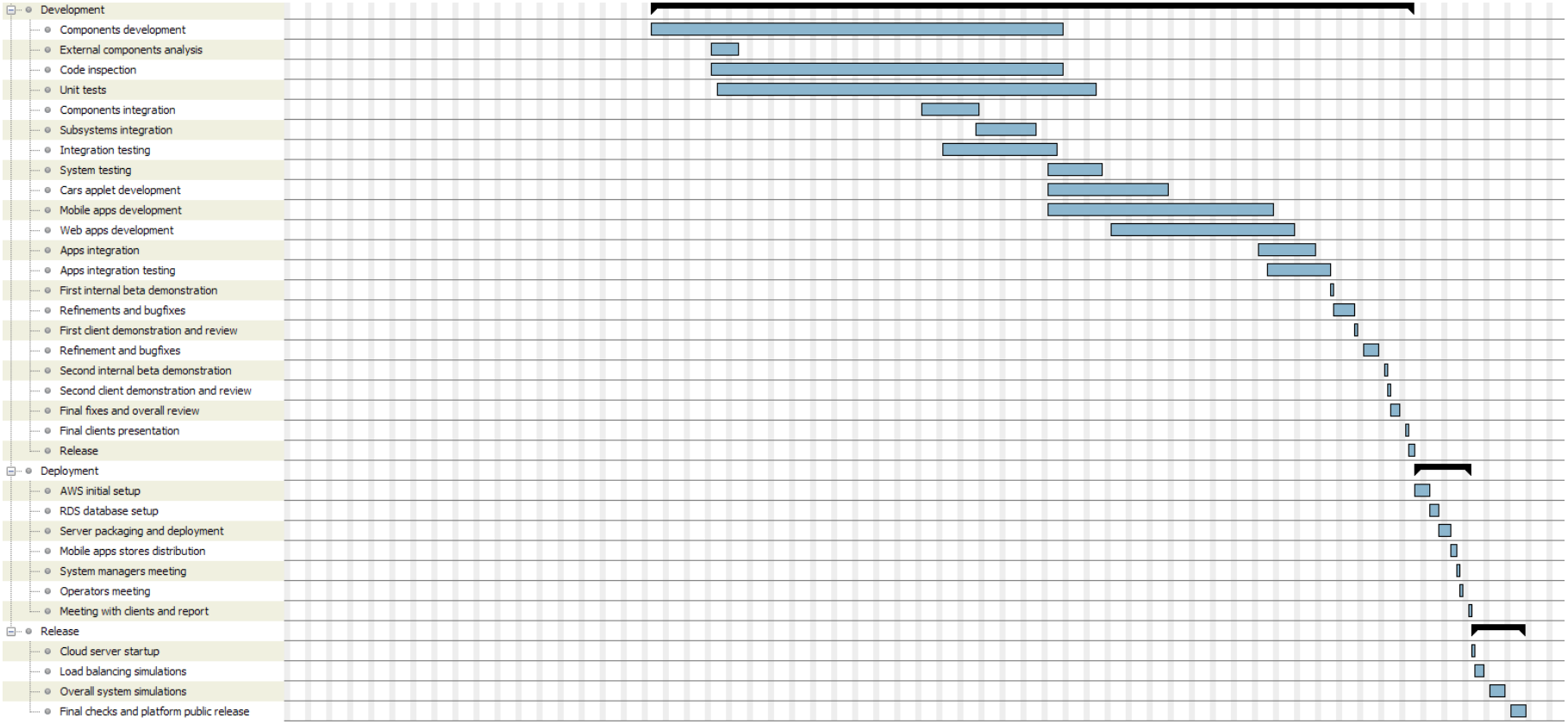
This schedule also takes into account weekends (gray vertical bars), so times assigned to the various tasks have been planned with realistic durations.

The schedule has been defined using a convenient Gantt diagram, following the classical notation. Times are assigned using a day scale (i.e. the minimum length of a blue bar is one day, and activities can only start and end at the beginning of a day).

We can realize that the estimated total duration of the project, including actual development and deployment, is close to the *Time estimation* made during the COCOMO II analysis: this schedule goes from October 16th, 2016 to November 18th, 2017, thus implying about 13 months of commitment. Section 2.2.4 contained an average estimation of about 14 months.

The diagram, split into parts for convenience, starts from the following page.


GANTT project
2016
2017
Ottobre
Novembre
Dicembre
Gennaio
Febbraio
Marzo
Aprile
Maggio
Giugno
Luglio
Agosto
Settembre
Ottobre
Novembre
Nome
Requirement Analysis and Specifications
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
RASD first draft
Goals detailed definition
Specifications detailed definition
Requirements detailed definition
Assumptions final definition
UI Mockups initial elaboration
Example scenarios definition
UML use case diagrams and descr. draft
Example alloy model analysis and worlds
Meeting with clients and review
RASD second draft
Final goals definition
Final specifications definition
Final requirements definition
Final assumptions definition
UI Mockups refinements
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
Platform Design
First high-level platform design
High level component analysis
Meeting with clients and review
DD first draft
Component structure design
Persistance design
Deployment design
Interfaces detailed definition
Tiers and layers definition
Algorithms drafts
Requirements traceability check
Meeting with clients and review
DD second draft
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation


Development
Components development
External components analysis
Code inspection
Unit tests
Components integration
Subsystems integration
Integration testing
System testing
Cars applet development
Mobile apps development
Web apps development
Apps integration
Apps integration testing
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
Deployment
AWS initial setup
RDS database setup
Server packaging and deployment
Mobile apps stores distribution
System managers meeting
Operators meeting
Meeting with clients and report
Release
Cloud server startup
Load balancing simulations
Overall system simulations
Final checks and platform public release

4. Resource allocation

In this section, we will provide a planning for the subdivision of the just scheduled tasks between the human resources available for our project, i.e. we ourselves. Data will be presented, like in section 3, using a properly formatted Gantt diagram.

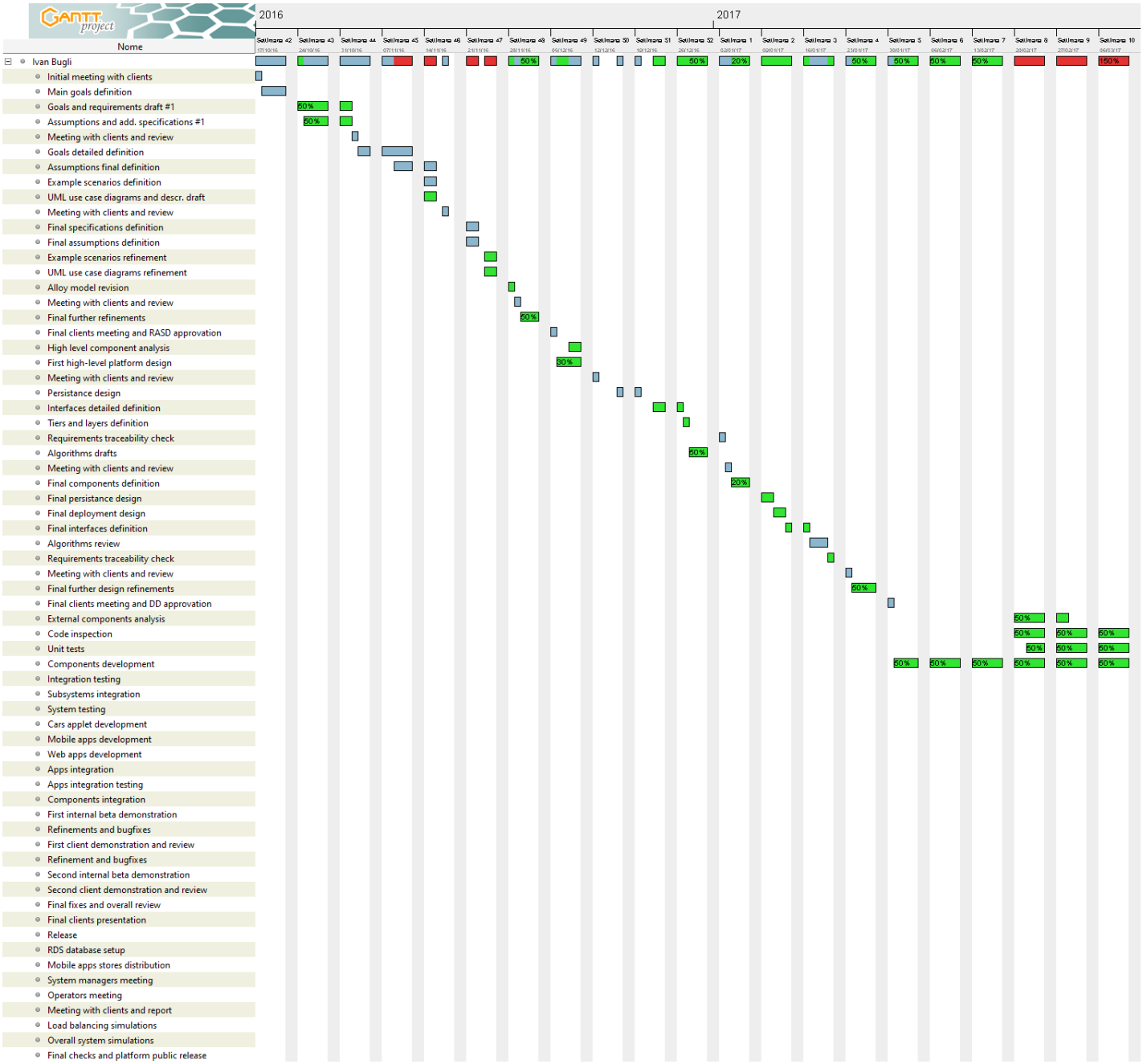
This diagram considers the fact that human resources are not supposed to work during weekends (the vertical grey bars), so these are always left free for both of us. Moreover, the diagram also highlights what part of every task each of us is meant to work on, in percentage terms. The blocks representing each task are colored to better put in evidence the percentage the one of us it's assigned to is supposed to complete. Plus, quite every percentage is written in the blocks themselves, if space allows it. The green blocks represent task assigned to a person for less than 100%, while red blocks represent tasks assigned contemporarily, resulting in a workload of more than 100%. If a task is assigned wholly to one of us only the corresponding block is instead azure.

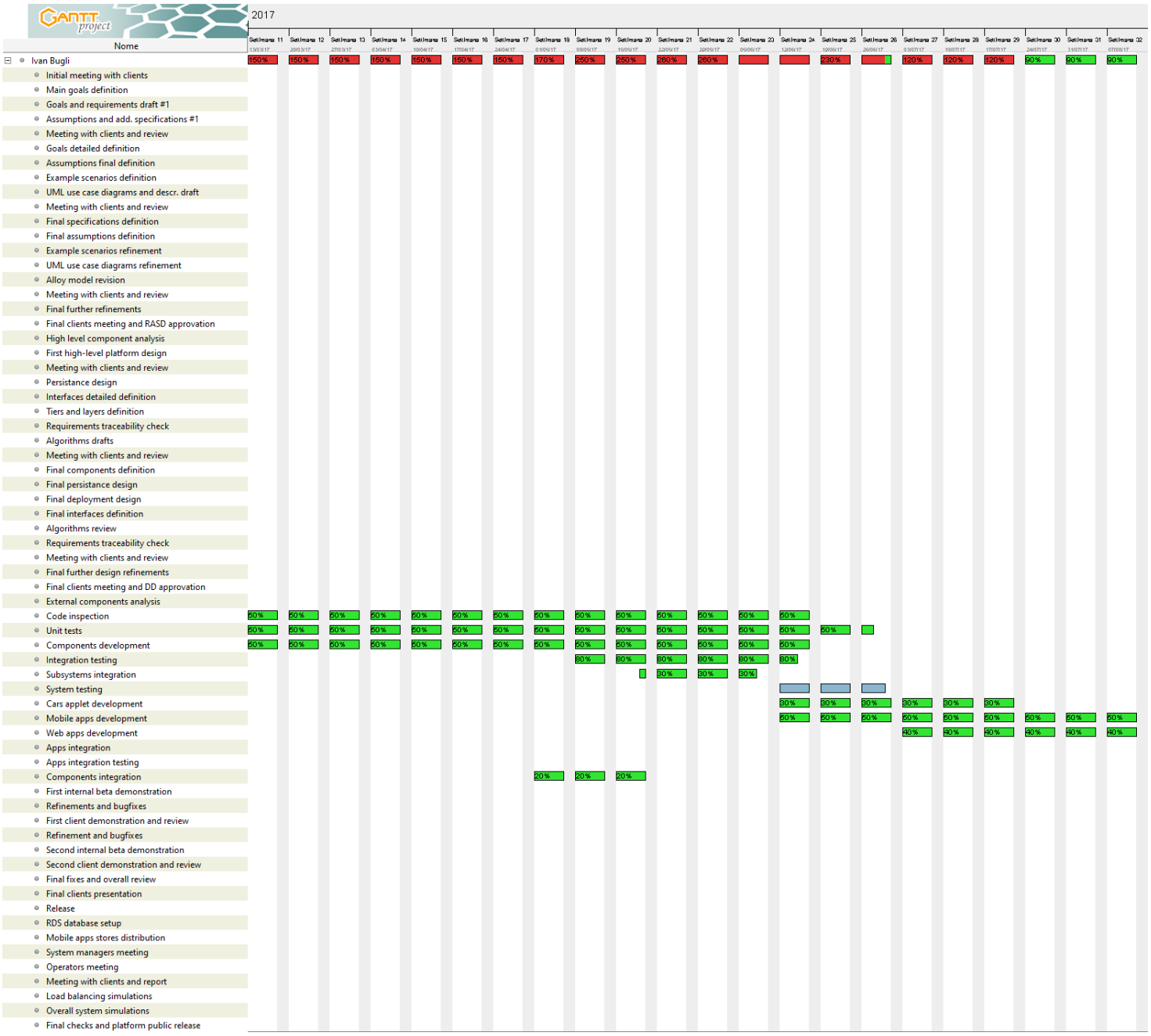
To correctly read the diagram is also important to note that the horizontal bar on top of the schedule represents the total workload for the person taken into account in the diagram from the start to the end of the whole project. It is the result, day by day, of the sum of every task this specific person has to do that day, considering the percentage of the tasks, too. For example, if during a certain day one is supposed to work on two task alone and on another one in terms of 30%, the total workload for that day would be 230%, resulting in a red block.

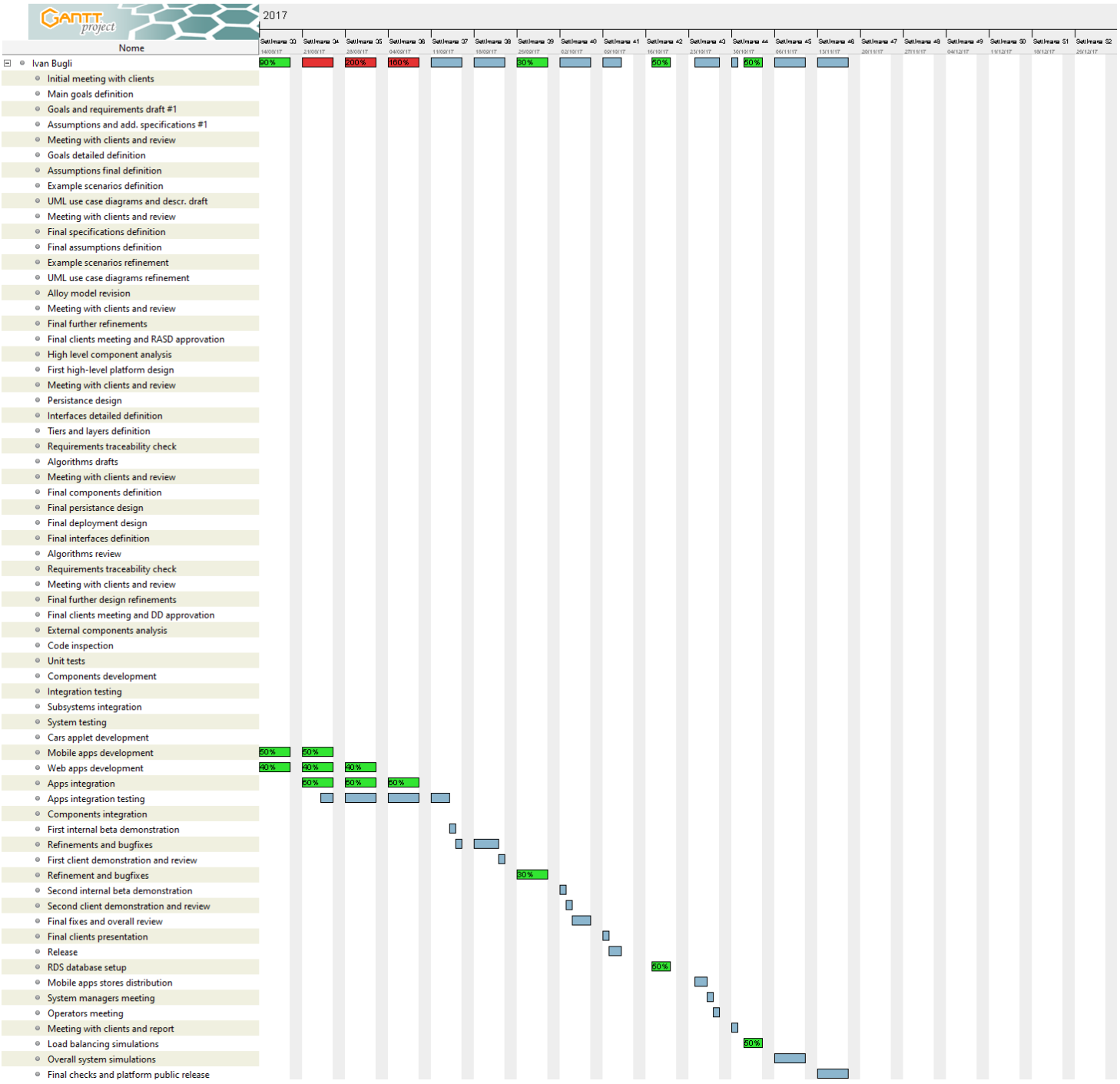
We made quite a precise analysis trying not to exceed a workload of 250% per day per person, distributing tasks in a smart way. Moreover, we assigned tasks also taking into account our attitudes and abilities, thus maximizing the expected final product result and quality.

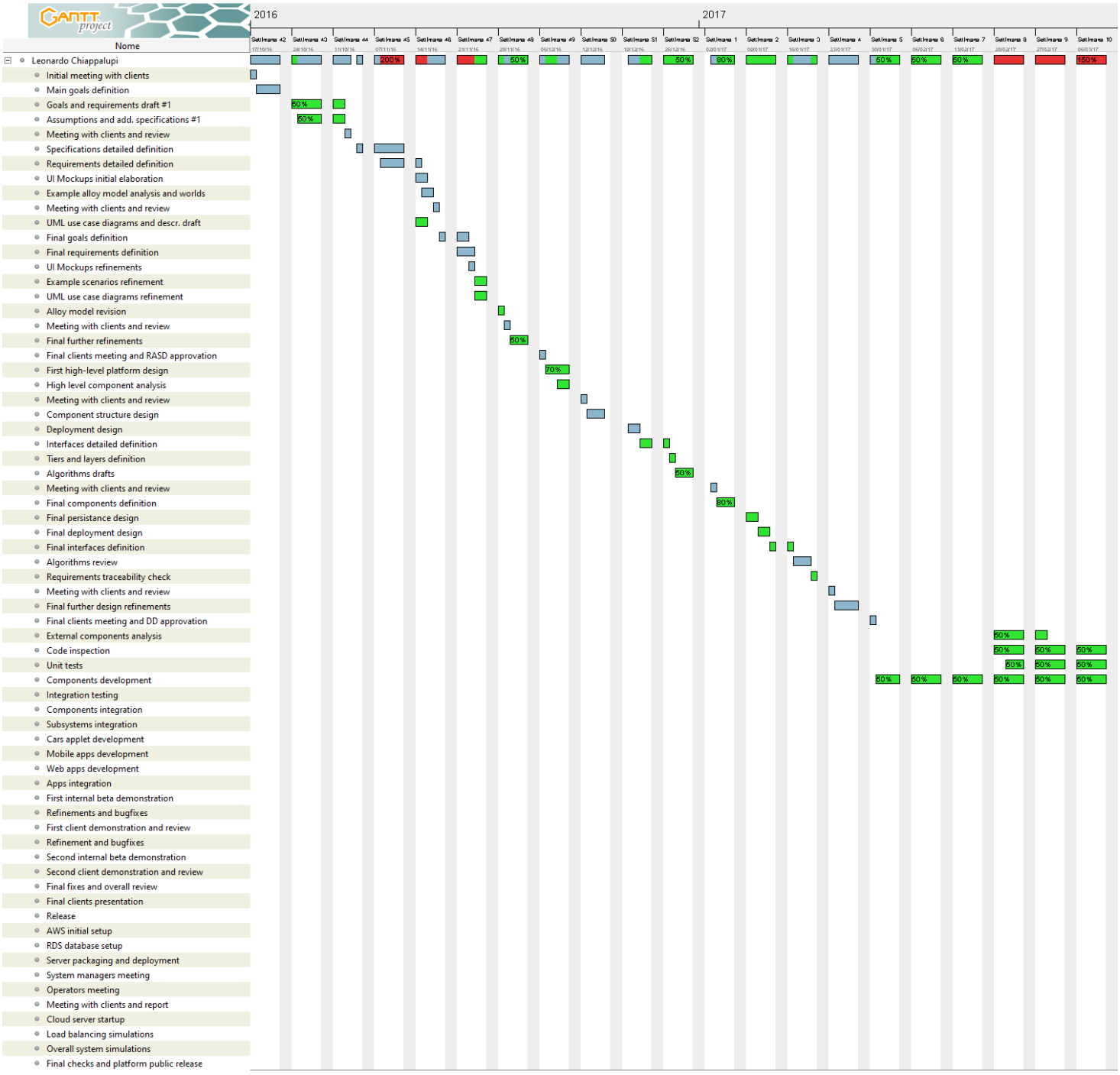
Lastly, we want to put in evidence the fact that some tasks are assigned to one person only, but other, as said before, are assigned to both of us. In this case, it can either be that the task is split into two exact part (50 and 50 percent), that the task is split unequally resulting however covered wholly (for example 30 and 70 percent) or that the task is split unequally but resulting covered for more than 100% (for example 100 and 50 percent); this means that our work overlaps and we are supposed to help each other or to check one another work, to improve the result of the task. There is one last possible case, that is when a task is assigned 100% to both of us; this means we have to do it completely together. An example are the meetings with the clients, which are always assigned to both of us contemporarily.

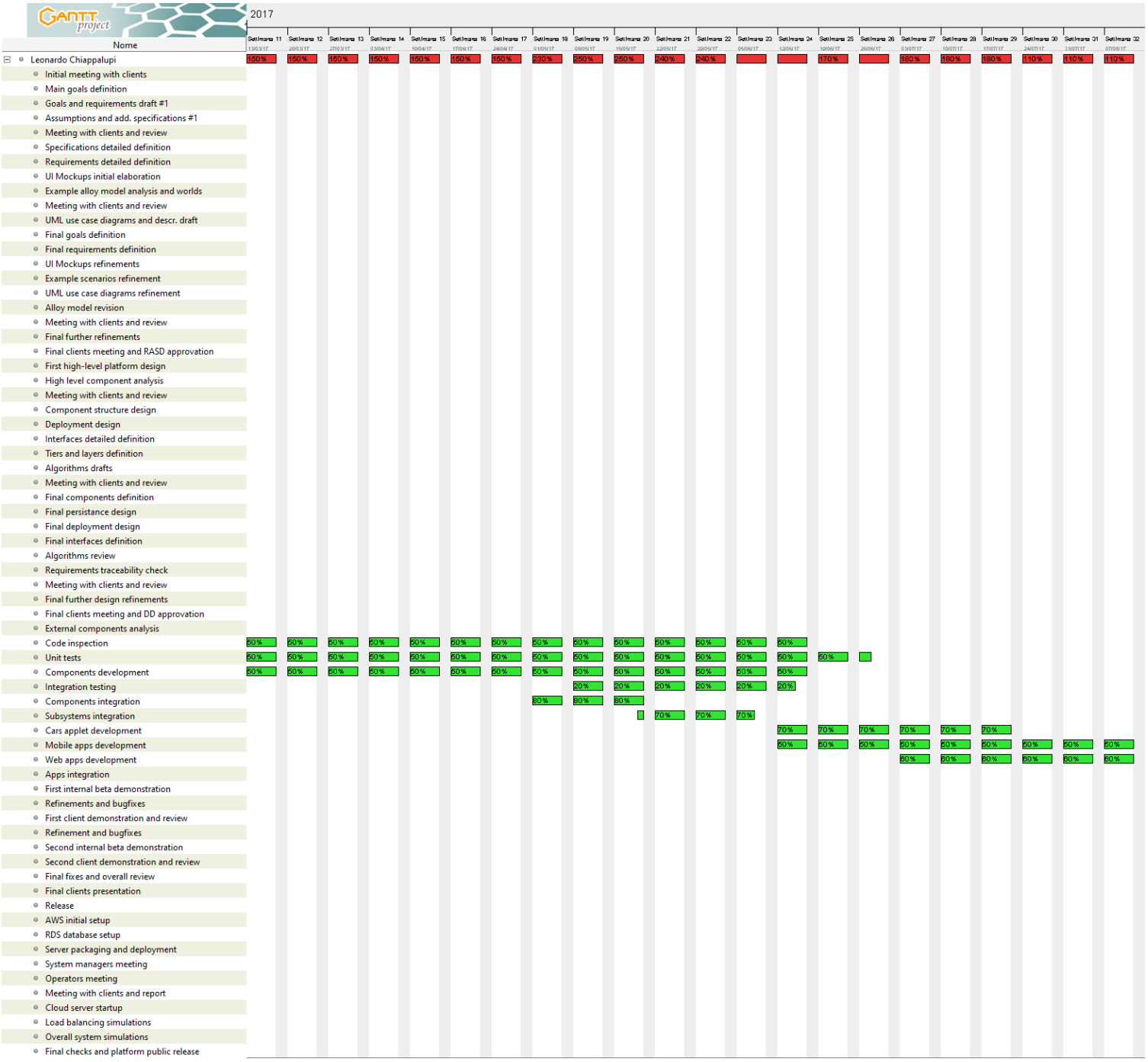
The diagrams, split into parts for convenience, start from the following page.

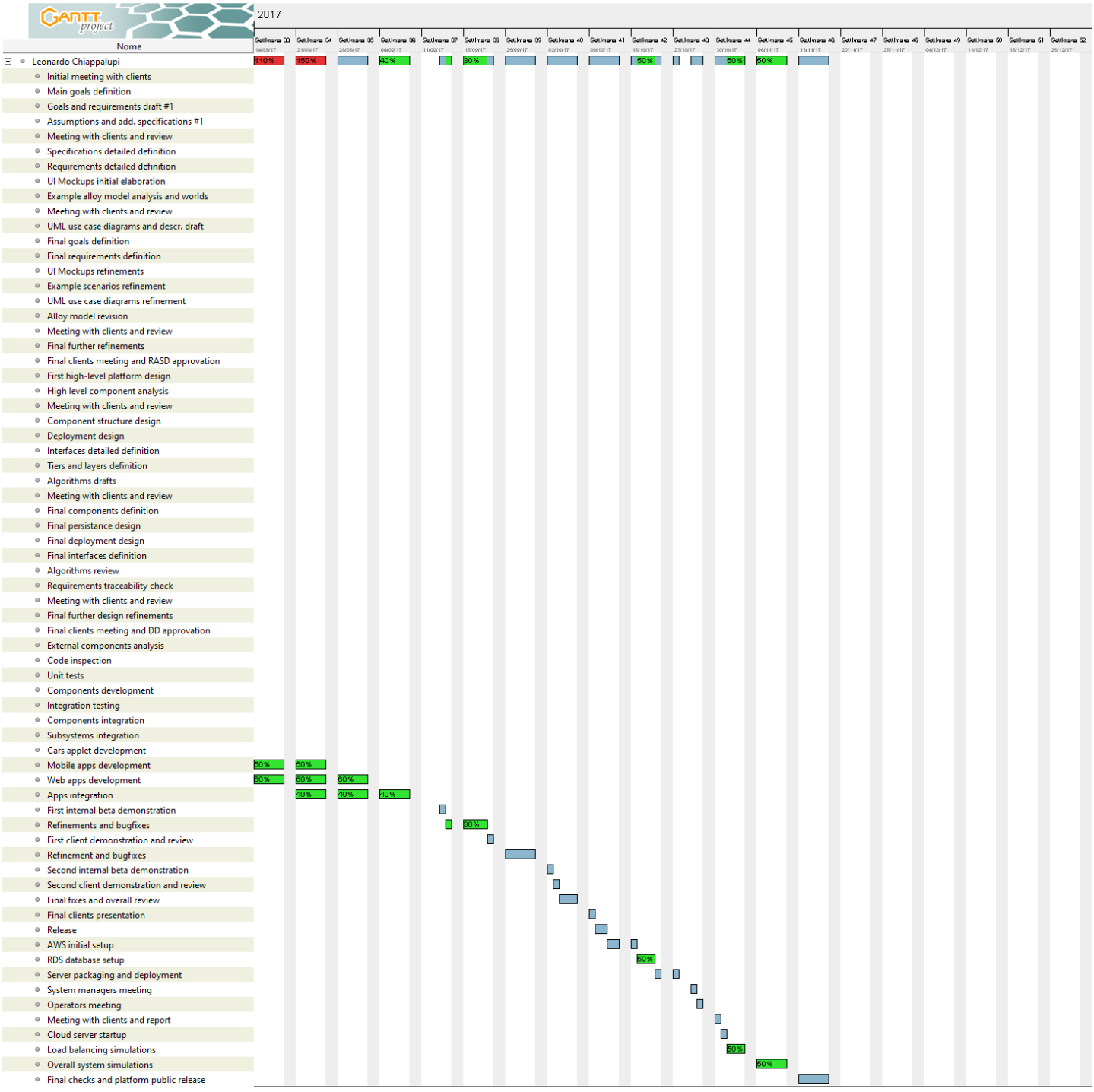

2016
2017
Nome
Ivan Bugli
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
Goals detailed definition
Assumptions final definition
Example scenarios definition
UML use case diagrams and descr. draft
Meeting with clients and review
Final specifications definition
Final assumptions definition
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
High level component analysis
First high-level platform design
Meeting with clients and review
Persistance design
Interfaces detailed definition
Tiers and layers definition
Requirements traceability check
Algorithms drafts
Meeting with clients and review
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation
External components analysis
Code inspection
Unit tests
Components development
Integration testing
Subsystems integration
System testing
Cars applet development
Mobile apps development
Web apps development
Apps integration
Apps integration testing
Components integration
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
RDS database setup
Mobile apps stores distribution
System managers meeting
Operators meeting
Meeting with clients and report
Load balancing simulations
Overall system simulations
Final checks and platform public release


GANTT project
2017
Nome
Ivan Bugli
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
Goals detailed definition
Assumptions final definition
Example scenarios definition
UML use case diagrams and descr. draft
Meeting with clients and review
Final specifications definition
Final assumptions definition
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
High level component analysis
First high-level platform design
Meeting with clients and review
Persistance design
Interfaces detailed definition
Tiers and layers definition
Requirements traceability check
Algorithms drafts
Meeting with clients and review
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation
External components analysis
Code inspection
Unit tests
Components development
Integration testing
Subsystems integration
System testing
Cars applet development
Mobile apps development
Web apps development
Apps integration
Apps integration testing
Components integration
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
RDS database setup
Mobile apps stores distribution
System managers meeting
Operators meeting
Meeting with clients and report
Load balancing simulations
Overall system simulations
Final checks and platform public release


GANTT project
2017
Nome
Ivan Bugli
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
Goals detailed definition
Assumptions final definition
Example scenarios definition
UML use case diagrams and descr. draft
Meeting with clients and review
Final specifications definition
Final assumptions definition
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
High level component analysis
First high-level platform design
Meeting with clients and review
Persistance design
Interfaces detailed definition
Tiers and layers definition
Requirements traceability check
Algorithms drafts
Meeting with clients and review
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation
External components analysis
Code inspection
Unit tests
Components development
Integration testing
Subsystems integration
System testing
Cars applet development
Mobile apps development
Web apps development
Apps integration
Apps integration testing
Components integration
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
RDS database setup
Mobile apps stores distribution
System managers meeting
Operators meeting
Meeting with clients and report
Load balancing simulations
Overall system simulations
Final checks and platform public release


GANTT project
2016
2017
Nome
Leonardo Chiappalupi
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
Specifications detailed definition
Requirements detailed definition
UI Mockups initial elaboration
Example alloy model analysis and worlds
Meeting with clients and review
UML use case diagrams and descr. draft
Final goals definition
Final requirements definition
UI Mockups refinements
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
First high-level platform design
High level component analysis
Meeting with clients and review
Component structure design
Deployment design
Interfaces detailed definition
Tiers and layers definition
Algorithms drafts
Meeting with clients and review
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation
External components analysis
Code inspection
Unit tests
Components development
Integration testing
Components integration
Subsystems integration
Cars applet development
Mobile apps development
Web apps development
Apps integration
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
AWS initial setup
RDS database setup
Server packaging and deployment
System managers meeting
Operators meeting
Meeting with clients and report
Cloud server startup
Load balancing simulations
Overall system simulations
Final checks and platform public release


GANTT project
2017
Nome
Leonardo Chiappalupi
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
Specifications detailed definition
Requirements detailed definition
UI Mockups initial elaboration
Example alloy model analysis and worlds
Meeting with clients and review
UML use case diagrams and descr. draft
Final goals definition
Final requirements definition
UI Mockups refinements
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
First high-level platform design
High level component analysis
Meeting with clients and review
Component structure design
Deployment design
Interfaces detailed definition
Tiers and layers definition
Algorithms drafts
Meeting with clients and review
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation
External components analysis
Code inspection
Unit tests
Components development
Integration testing
Components integration
Subsystems integration
Cars applet development
Mobile apps development
Web apps development
Apps integration
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
AWS initial setup
RDS database setup
Server packaging and deployment
System managers meeting
Operators meeting
Meeting with clients and report
Cloud server startup
Load balancing simulations
Overall system simulations
Final checks and platform public release


GANTT project
2017
Nome
Leonardo Chiappalupi
Initial meeting with clients
Main goals definition
Goals and requirements draft #1
Assumptions and add. specifications #1
Meeting with clients and review
Specifications detailed definition
Requirements detailed definition
UI Mockups initial elaboration
Example alloy model analysis and worlds
Meeting with clients and review
UML use case diagrams and descr. draft
Final goals definition
Final requirements definition
UI Mockups refinements
Example scenarios refinement
UML use case diagrams refinement
Alloy model revision
Meeting with clients and review
Final further refinements
Final clients meeting and RASD approvation
First high-level platform design
High level component analysis
Meeting with clients and review
Component structure design
Deployment design
Interfaces detailed definition
Tiers and layers definition
Algorithms drafts
Meeting with clients and review
Final components definition
Final persistance design
Final deployment design
Final interfaces definition
Algorithms review
Requirements traceability check
Meeting with clients and review
Final further design refinements
Final clients meeting and DD approvation
External components analysis
Code inspection
Unit tests
Components development
Integration testing
Components integration
Subsystems integration
Cars applet development
Mobile apps development
Web apps development
Apps integration
First internal beta demonstration
Refinements and bugfixes
First client demonstration and review
Refinement and bugfixes
Second internal beta demonstration
Second client demonstration and review
Final fixes and overall review
Final clients presentation
Release
AWS initial setup
RDS database setup
Server packaging and deployment
System managers meeting
Operators meeting
Meeting with clients and report
Cloud server startup
Load balancing simulations
Overall system simulations
Final checks and platform public release

5. Risk management

In this section, we will go through an extensive list of risks that Power EnJoy may face during its lifecycle. For every possible danger, we will also provide a possible solution, trying to figuring out whatever problem there could be with the platform – from technical ones to political or environmental ones.

Risk	Solution or mitigation
Changes in national road rules At a certain point of times, laws and rules concerning the circulation of (electrical) vehicles, licenses, etc. may change in an unpredictable manner that may disrupt Power EnJoy service.	*Assign to some members of the management team the duty of being constantly aware of possible laws concerning fields Power EnJoy is interested into; these people will be also in charge of designing backup plans in advance to quickly solve the problem.*
Vandalism, abuses on the service There is a risk of our service being highly misused or subject to vandalism, especially during night. This could lead to sudden deficiencies of cars and to the inability for the service to run.	*Maintain a constant stock of additional cars to replace the ones that will take a long time to fix; consider hiring a private security service to provide surveillance in the most dangerous zones.*
Changes in parking types and plug parking areas Parking areas are not decided by us in terms of location or type; the city government may make sudden changes and Power EnJoy may face inconsistencies with the parking areas.	*Establish a direct line with the department of transportation for the city, asking to be reported in advance with possible modifications related to parking areas; we designed the system so that editing them is very straightforward.*
Car theft There could be the chance of cars being stolen or taken abroad; this would result in high economical losses for the client.	*Consider equipping cars with an additional GPS module to find them when stolen; see also solutions to Vandalism, abuses on the service*
Similar services competitors Car sharing, even for electric cars, already has several companies active and in a high competition market: even if we provide unique features, there is the risk that customers will prefer the safety and the probable higher coverage of more renowned companies.	*Advertise the platform a lot, stressing out the unique features Power EnJoy may provide; start with very cheap and affordable fees, maybe including a free time window when subscribing.*
Changes in external systems Since we rely upon several external components (Google Maps, the PaymentHandler – PayPal and/or a bank, the DBMS, etc.), significant changes in their APIs or features may make the	*Predispose backup alternative providers for every service: for instance, using Open Street Map instead of Google Maps. The development of the system should take this possibility into consideration to allow a quick switch in case of problems.*

service unstable or unusable all of a sudden.	
Amazon AWS technical issues Amazon Web Services is our cloud platform of choice for the system; however, some of the services we are going to use maybe shut down in the future, or just completely upset. This may result in an instantaneous inability of deliver our service.	*Such changes are usually communicated in advance to the main AWS account owner; thus, our team must remain heedful about e-mails from Amazon about services changes. To stay completely safe, we suggest to maintain a low-cost service backup on another platform, in case of emergencies.*
Amazon AWS billing issues Amazon Web Services provides a pay-as-you-go model: changes in the fees or an enormous increase of users may lead to an unsustainable model on the long term (talking about money).	*Negotiate in advance with other cloud platforms providers when an increase in the number of users is noticed, so to have alternatives ready when Amazon is not convenient anymore.*
Security threats Even if we are relying upon AWS own security systems, and payments are secured by the external payment handler, we can still have personal information thefts and/or phishing, which may have bad consequences for our customers and from a legal point of view, for the company itself.	*Consider hiring a network security consulting agency for maintaining the security of exposed Power EnJoy modules; sensitize users via email about risks of phishing etc.*
Lack of critical workers For the system as it has been designed by us, it is crucial to have a minimum number of both system managers and operators to constantly be ready to activate and monitor their devices. An improvise lack of the minimum amount of workforce may lead to critical service shortages.	*Consider hiring additional system managers and workers and to rotate them constantly, maybe also asking them to be ready to replace some colleagues when there are deficiencies in the personnel.*
Loss of core application source code A remote risk of failure is to have a significant loss of source code of the application, either for the core backend or for the mobile/web applications.	*Introduce a mandatory versioning system for all developers to synchronize source code and push it to remote repositories, in order to always maintain cloud backup copies.*
Troubles with cellular services used for cars Changes in the subscription plans of mobile data providers for cars, or in their coverage/level of service, may severely impact the vendibility of Power EnJoy.	*Consider making arrangements in advance with an alternative mobile carrier, or at least to decide a feasible secondary plan, to have a reliable choice in case of prolongated issues.*

6. Appendices

6.1 Tools used

- Microsoft Office Word 2016
  https://products.office.com/it-it/home

  For redacting, reviewing, layout and graphic design of this document

- GanttProject 2.8.1
  http://www.ganttproject.biz/

  For the elaboration of the schedule and resources allocation

6.2 Effort spent

The present document required almost the same time for both the curators (Leonardo Chiappalupi, Ivan Bugli).

The total time spent **apiece** on the creation of the paper is: ~16 Hours.

We will also summarize here the total time spent **by each curator** on all the documents concerning this project:

- RASD: ~45h
- DD: ~35h
- ITPD: ~20h
- PP: ~16h

The total time spent apiece on the project, therefore, is: ~116h.

6.3 References

- **QUALITATIVE SOFTWARE MANAGEMENT, FUNCTION POINTS LANGUAGE TABLE - VERSION 5:** http://www.qsm.com/resources/function-point-languages-table
- **USSC/CSSE COCOMO® II REFERENCE RESOURCES:** http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html
- **INTERNATIONAL FUNCTION POINTS USERS GROUP:** http://www.ifpug.org/
- **POWER ENJOY: REQUIREMENT ANALYSIS AND SPECIFICATION DOCUMENT – CHIAPPALUPI L., BUGLI I.** Politecnico di Milano, 2016
- **POWER ENJOY: DESIGN DOCUMENT – CHIAPPALUPI L., BUGLI I.** Politecnico di Milano, 2016
- **POWER ENJOY: INTEGRATION TEST PLAN DOCUMENT – CHIAPPALUPI L., BUGLI I.** Politecnico di Milano, 2016