POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
M. Sc. in Computer Science and Engineering
Dipartimento di Elettronica, Informatica e Bioingegneria

Power EnJoy

Software Engineering 2 - Project

Project Plan

Version 1.0

Authors:
Andrea BATTISTELLO Matr: 873795
William DI LUIGI Matr: 864165

Academic Year 2016 - 2017

1 Introduction

1.1 Purpose

The purpose of the project plan document is to estimate first the size, measured in Source Lines of Code, of the project to be developed and to estimate next the overall duration (thus the cost). The estimation in the first two points is done with the use of function points approach and COCOMO II.
Once we have estimated the duration of the project, we can then schedule the activities and resources during that period to have a clear idea about deadlines, milestones and tasks. Finally, the PP document takes in consideration multiple risk factors that can determine the failure of the project and comes with a strategy to prevent and/or resolve such problems.

1.2 Scope

Duckburg is a very large city, with a lot of people. Over the years, the number of people with a driving license has increased. This caused a lot of traffic, boosted air pollution, and made it really hard to find a parking spot.

The purpose of Power EnJoy is to provide an alternative to this. We want to create a commuting service that is highly available and reliable. In fact, the reason why people drive instead of using public transport is usually related to the fact that they don't want to wait for a bus / train (low availability) and they don't want to risk not getting home if there's no public transport available (low reliability) because of a strike, vehicle malfunctions, or just because it's late in the night.

This system will allow users to rent a car for a period of time. PE will let anyone see the available cars (by showing in a map those cars that are closer to the user's location). Moreover, PE will let activated users actually book those cars, and will charge them according to the duration and usage (discounts will apply in specific cases). The system will also have an administrative interface, which will not be accessible by standard users.

1.3 Definitions, acronyms, abbreviations

1.3.1 Definitions

- **Payment information** refers to either a bank account or a credit card number.
- **Device** indicates either a mobile phone / tablet running PEM or a browser that is using PWA.
- **Safe area** refers to any legal parking spot in the city, where any driver can leave the car without interfering with the traffic.
- **PE Safe area** is a safe area that can be used exclusively by PE cars.
- **Web server** is a system that receives HTTP requests and provides web-based contents.

- **Application server** is a system that contains all the business logic. Exposes some interfaces that are used by the web server, mobile applications and cars. It uses containers to balance the requests load.
- **Component** is a piece of software that encapsulates some functionalities and can be reused
- **Subsystem** is a logical and physical subdivision of the system (e.g. the Application server, the Android app, the iOS app, the car app and the admin web app are all different subsystems of the same system).
- **Container** is a piece of software that manages the components and offers an abstraction for parallelism, transactions and many others.
- **Entity** is a class that maps directly into the database. An example is given by JPA.

1.3.2 Acronyms

- **PE** Power EnJoy system
- **PEM** Power EnJoy Mobile application
- **PEW** Power EnJoy Web application
- **PEA** Power EnJoy Administrator application
- **PEC** Power EnJoy Car application
- **SMS** Short Message Service
- **GPS** Global Positioning System
- **RASD** Requirement analysis and specification document
- **DD** Design document
- **UI** User interface
- **UX** User experience design
- **MVC** Model view controller
- **JDBC** Java DataBase Connectivity
- **JPA** Java Persistence API
- **JEE** Java Enterprise Edition
- **EJB** Enterprise Java Bean
- **SLOC** Source Lines of Code

1.3.3 Abbreviations

- **[Gn]** n-th goal
- **[Dn]** n-th domain assumption
- **[Rn.m]** m-th requirement related to goal [Gn]
- **[Sn]** n-th subsystem
- **[SnIm]** m-th integration for the n-th subsystem
- **[SnIm-Tk]** k-th test of the integration above
- **[SIn]** n-th integration between subsystems
- **[SIn-Tk]** k-th test for the integration above

1.4 Reference documents

- [1] IEEE Software Engineering Standards Committee, “29148-2011 - Systems and software engineering — Life cycle processes — Requirements engineering”, 2011.

- [2] Payment Card Industry (PCI) Data Security Standard, v3.2, PCI Security Standards Council, LLC.
- [3] Response Times: The 3 Important Limits, JAKOB NIELSEN, 1993.
- [4] Software Engineering 2 course slides.
- [5] The assignment of *Power EnJoy*
- [6] RASD (Requirement specification and Specification document) of *PowerEnjoy*
- [7] DD (Design document) of *PowerEnjoy*
- [8] COCOMO II Model definition manual, ver 2.1

1.5 Document structure

This document is composed of five sections and an appendix:

- The **introduction** section, this one, is intended to find the goal of a design document, clarifies the definitions and acronyms used throughout the document and gives a general idea of the main functionalities of the system to be developed.
- The **size estimation** section will make use of the function points to estimate the expected number of source lines of code of the project.
- The **cost and effort** section will start from the size of the project estimated in the previous section and applies COCOMO II approach to estimate the effort and the time to develop. This approach will take into account different scale factors and cost drivers that must be tailored for this project.
- The **tasks and schedule** section will define the tasks, milestones and deliverables to be produced and uses the previous informations to plan the amount of work over time.
- The **resource allocation** section will assign every resource to the tasks they should work on.
- Finally, the **risks and recovery actions** section will list a possible set of risks that may occur and defines a strategy to prevent or - if they become real - fix them.

2 Size estimation

In this section we will estimate the size of the PowerEnjoy system using the function points approach. Estimating the size and cost of a software project is not a trivial task. There are many variables that can influence the development of a software system, so usually we rely on the past experience of the manager.
However there are some algorithmic methods to estimate such cost by taking into account different aspects of the software system. In this project we will adopt the function point approach developed by Allan Albrecht in 1965 to estimate the size of the project, and then the COCOMO method to estimate the cost.

2.1 Function points

The function points approach assigns a certain number of function points (FP) based on the amount of functionalities to be developed and their complexity.

The estimation is based on 5 different types of functionalities:

- **Internal Logical File (ILF)**: homogeneous set of data used and managed by the application.
- **External Logical File (ELF)**: homogeneous set of data used by the application but generated and maintained by other applications.
- **External Input**: elementary operation to elaborate data coming from the external environment.
- **External Output**: elementary operation that generates data for the external environment; it usually includes the elaboration of data from logical files.
- **External Inquiry**: elementary operation that involves input and output, without significant elaboration of data from logical files.

Each of them has an associated weight depending on the complexity of the functionalities to be implemented. The weights of each type of functionality is represented in the table below:

Function type	**Complexity**		
	Low	**Average**	**High**
Internal Logical File	7	10	15
External Logical File	5	7	10
External Input	3	4	6
External output	4	5	7
External Inquiry	3	4	6

2.2 Internal Logical File

PE system must have access to the following internal files in the database:

- **User information**; especially: name, address, phone number, email, driving license number, ID number, password and payment information. The complexity of this ILF is high, because:
    - This data is user-generated. As such, it needs to be checked and validated and, if necessary, invalidated (e.g. if a user entered an ID number which belongs to another person with the same name and last name).
    - All of these fields are subject to change (the user might want to change them, or they might have a limited validity, like a driving license), so it's crucial to implement a way to change them.
    - The user might forget the password to access the system. The easy solution would be to just send them back their password, however, that would violate the PCI Data Security Standard. Thus, it's crucial to implement a system to restore user credentials (which includes some kind of identity verification).
- **Car positions and status**; in particular: the GPS coordinates, battery status, plate number and a series of events related to the car (e.g. unlock, start driving, stop driving events). This data is machine-generated, so we can expect it to always be well-formed. The complexity of this ILF is average because, since this data is generated at a quite high constant rate, the amount of records to be stored and analyzed (e.g. by maintaining indices on the database) will also be quite high. Special attention must be paid in order to manage this data correctly.
- **Ride information**; the following information is required: user ID number, initial time, final time, car ID number and a sequence of samples that stores information about the position of the car and its battery level taken during the entire trip duration. It is also important to store information about the number of passengers, the final battery level and whether the driver plugged the battery in order to correctly calculate the trip cost. The complexity of this ILF is average because this information is mission-critical in delivering discounts (e.g. a discount for users that use the car with their friends) so special attention must be paid to avoid users gaming the system to obtain unfair discounts. For example, we could exploit the trigger system of the database to set up "alarms" to signal unusual updates or data right away, and then have a supervisor look into the matter.
- **Payment bills and credit card number**; for registered users. The complexity of this ILF is average because this is sensitive data which needs special care and attention when handled.
- **Reservations**; in particular the date and the car id number. This data is machine-generated (at the request of registered users) so it's well formed. The complexity of this ILF is low.

The following table summarizes the FP for this section:

Internal Logical File	**Complexity**	**FPs**
User information	High	15
Car position and status	Average	10
Ride information	Average	10
Payment bills	Average	10
Reservations	Low	7
Total		52

2.3 External Logical File

PE system must interact with several external interfaces and resources. We can identify the following external logical files:

- **Governmental database**; PE must access this database through RESTful API calls in order to validate driving license numbers and ID card numbers. This interaction only needs to make simple calls, thus requiring low complexity.
- **Bank**; this is required by the payment manager to charge the users and to check the validity of payment methods. This interaction must be carried out carefully in order to have reliable transactions, to avoid information leakage and to ensure that the assets aren't damaged, so it requires high complexity.
- **Geolocalization APIs**; this is used to provide address resolution service (given an address, return the corresponding GPS coordinates) and map services (show the map of a specified area). Since maps aren't static objects, the APIs to handle them are prone to changes and updates to the underlying data. Special attention should be paid in order to handle map updates, thus it requires low complexity.
- **SMS and phone service**; this provides the ability to send and receive SMS and to make calls from admin panel. Special attention should be paid to the limitations of the SMS system e.g. by validating the max length of the text and the characters used.

The following table summarizes the FP for the External Logical Files:

External Logical File	**Complexity**	**FPs**
Governmental database	Low	5
Bank	High	10
Geolocalization APIs	Low	5
SMS and phone service	Average	7
Total		22

2.4 External Input

PowerEnjoy takes several external inputs from different subsystems (PEM, PEW, PEA). The interactions are as follows:

2.4.1 User inputs

- **Login and logout**: this is a standard functionality and there are plenty of ready to use implementations. However, it's not a straightforward functionality as it needs to handle sessions / cookies (with all the security implications) so this would be average complexity.
- **Registration**: this is a rather complex process because it involves several checks on the validity of input data (driver license, ID card number, payment method) and also it involves the verification of the phone number via SMS. For these reasons it will be labeled as high complexity.
- **Account management**, that includes the possibility of edit and/or update basic profile information (name, password, email, phone number and so on), but it also involves sensitive information, such as payment method. Since this functionality must guarantee the consistency of data after any update, it is labeled as average complexity.
- **Unlock car, start driving** and **stop driving**: these operations are required in order to properly ride a car. For these functionalities we expect an average complexity because they need to interact also with PE Car and are not well-known operations.
- **Search cars**: this functionality may be of low complexity by itself, but since we have opted for an efficient way of retrieving the car positions (look at the DD [7] for reference), we expect the implementation of this algorithm take average complexity.
- **Reserve car**: this functionality informs the car about the reservation and takes care that the reservation does not expire (otherwise a fine would be charged to the user). We expect this to be average complexity because many users can do reservations at the same time, thus the implementation requires avoiding any concurrent issue.
- **Cancel reservation**: this will be labeled as low complexity since it only have to release the lock on the car.
- **Pay**: this functionality only need to inform the bank about the amount to charge, but since money transactions are critical, we allow this component to have an average complexity.

2.4.2 Administrator input

- **View reservation history**: this functionality only need to access the database, so we labeled it with low complexity
- **Call driver** and **notify maintenance team**: these functionalities are labeled with low complexity since their only purpose is to forward a function call.
- **View available cars**: this is the same functionality as above, with the addition that the call center operators can visualize more details. As the above functionality, this requires average complexity.

- **Reimburse**: this functionality need the interaction with the bank, so this requires an average effort.

The following table summarizes the FP for External Inputs:

External Input	**Complexity**	**FPs**
Login and logout	Average	4
Registration	High	6
Account management	Average	4
Unlock car	Average	4
Start driving	Average	4
Stop driving	Average	4
Search cars	Average	4
Reserve car	Average	4
Cancel reservation	Low	3
Pay	Average	4
View reservation history	Low	3
Call driver	Low	3
Notify maintenance team	Low	3
View available cars	Average	4
Reimburse	Average	4
Total		58

2.5 External Output

The external outputs produced by PowerEnjoy are:

- **SMS confirmation** message, used to confirm the phone number inserted by the user in registration phase.
- **Password** generated by PowerEnjoy during registration, as stated in the requirements analysis [6]
- **Email notification** to maintenance team, to inform them about some cars to be repaired.
- **Phone calls** done by call center operators when they need to call the driver for any reason.

External Output	**Complexity**	**FPs**
SMS confirmation	Average	5
Password generated	Low	4
Email to maintenance team	Low	4
Phone calls	Low	4
Total		17

2.6 External Inquiry

The external inquiries we have identified are:

- **Search for cars**, as it has to fetch the position of all the cars and return only the ones within a specified area. This gives it an average complexity.
- **Reservation history** of users, done by call center operators. This operation needs to access and elaborate many reservation entries from the database.

These operations sums up in the following table:

External Inquiries	**Complexity**	**FPs**
Search car	Average	4
Reservation history	Low	3
Total		7

2.7 Overall estimation

Summing all up, we have the following function points for every function type described above:

Function type	**FPs**
Internal Logical File	52
External Logical File	22
External Input	58
External output	17
External Inquiry	7
Total FPs	156

We can now estimate the number of SLOC. We have chosen in the DD [7] that the development platform is JEE. So we can adjust the FPs according to this factor:

Language	Average	Median	Low	High
JEE	46	49	15	67

Since we don't know exactly how the complexity of JEE implementation would be, we can estimate an average cost factor, thus giving us:

$$SLOC = 156\,FP * 46\,\frac{SLOC}{FP} = 7146\,SLOC$$

with an upper bound of:

$$SLOCmax = 156\,FP * 67\,\frac{SLOC}{FP} = 10452\,SLOC$$

3 Cost and effort estimation

In this section we will adopt COCOMO II to estimate the cost and effort of PowerEnjoy system.
In COCOMO II the effort is expressed in Person Months (PM). In order to overcome all the possible situations, the COCOMO effort equation takes into account several scale factors and cost drivers, that are accurately tuned by considering the actual cost of several past projects.

We will start from the scale factors and then we will consider the cost drivers. At the end of this section we will plug them in the effort equation to estimate the time needed to develop this project

3.1 Scale factors

Scale factors take into account economic, organizational and technical aspects of the environment in which the project is being developed. There are five scale factors to consider:

- **Precedentedness** (PREC): reflects the previous experience of the organization with this type of project. Very low means no previous experience, Extra high means that the organization is completely familiar with this application domain.
  Since we have no previous experience with this type of project, it will be low.

- **Development flexibility** (FLEX): reflects the degree of flexibility in the development process. Low value means that development process is fixed and can't be changed, high value means that the client have only set general goals.

- **Risk resolution** (RESL): reflects the extent of risk analysis carried out. Low value means no analysis, while higher value means that a specific and detailed analysis has been carried out.

- **Team cohesion** (TEAM): reflects how well the development team know each other and work together. Very low means very difficult interactions, Extra high means an integrated and effective team with no communication problems.

- **Process maturity** (PMAT): reflects the process maturity of the organization.

Here is the table that provides the coefficients for every scale factor defined above:

Scale factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented **6.20**	largely unprecedented **4.96**	somewhat unprecedented **3.72**	generally familiar **2.48**	largely familiar **1.24**	thoroughly familiar **0.00**

FLEX	rigorous **5.07**	occasional relaxation **4.05**	some relaxation **3.04**	general conformity **2.03**	some conformity **1.01**	general goals **0.00**
RESL	little (20%) **7.07**	some (40%) **5.65**	often (60%) **4.24**	generally (75%) **2.83**	mostly (90%) **1.41**	full (100%) **0.00**
TEAM	very difficult interactions **5.48**	some difficult interactions **4.38**	basically cooperative interactions **3.29**	largely cooperative **2.19**	highly cooperative **1.10**	seamless interactions **0.00**
PMAT	Level 1 Lower **7.80**	Level 1 Upper **6.24**	Level 2 **4.68**	Level 3 **3.12**	Level 4 **1.56**	Level 5 **0.00**

The following table will summarize our decisions of scale factors, as specified above in more details.

Scale factor	**Factor**	**Value**
PREC	Low	4.96
FLEX	High	2.03
RESL	Nominal	4.24
TEAM	Very high	1.10
PMAT	Low	6.24
Total		18.57

3.2 Cost drivers

COCOMO II specifies several cost drivers, that may vary depending on whether it is early design or post-architecture. Since we have already defined the architecture of the system in the Design Document [7] we will use the post-architecture cost drivers to have a more precise estimation of the cost.
The cost drivers are the following, as well specified in [8]:

3.2.1 Product factors

- **Required Software Reliability (RELY)**

This is the measure of the extent to which the software must perform its intended function over a period of time. If the effect of a software failure is only slight inconvenience then RELY is very low. If a failure would risk human life then RELY is very high.

RELY Descriptors:	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
Rating level	Very Low	Low	Nominal	High	Very High	Extra High
Effort multiplier	0.82	0.92	1.00	1.10	1.26	n/a

We estimate that for our project requires High reliability because we expect that our service will be highly used during strikes or big events. In these situations a failure can imply a consistent financial loss.

- **Database Size (DATA)**

This cost driver attempts to capture the effect large test data requirements have on product development. It is calculated as D/P: the number of bytes in the testing database over the SLOC in the program.

DATA Descriptors:		D/P < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
Rating level	Very Low	Low	Nominal	High	Very High	Extra High
Effort multiplier	n/a	0.90	1.00	1.14	1.28	n/a

We have estimated a size of roughly 3~4MB of overall test data, that brings us in the High rating level

- **Product Complexity (CPLX)**

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. The multiplier is then obtained with a subjective weighted average over these five areas.

CPLX						
Rating level	Very Low	Low	Nominal	High	Very High	Extra High
Effort multiplier	0.73	0.87	1.00	1.17	1.34	1.74

We have considered these rating level for every area:

- *Control operations*: High because we have real time map visualization
- *Computational operations*: Nominal, we do not require heavy computations.
- *Device-dependent operations*: Nominal, we don't have input/output intensive systems
- *Data management operations*: High, we have to store sensitive informations and the data are structured in a way to increase performance

- *User interface management operations*: Nominal, we don't have any particular user interface requirement.

Giving equal weight on all five sections, we have a CPLX factor of 1.102

- **Developed for reusability (RUSE)**

This factor takes into account the effort needed to construct components intended for reuse on current or future projects by creating more generic design of software and testing.

RUSE Descriptors:		none	across project	across program	across product line	across multiple product lines
Rating level	Very Low	Low	Nominal	High	Very High	Extra High
Effort multiplier	n/a	0.95	1.00	1.07	1.15	1.24

Since we have only one product, we require reusability across program (High RUSE).

- **Documentation Match to Life-Cycle Needs (DOCU)**

This cost driver takes into account the required documentation level. We opted for Nominal documentation that is good enough for this project.

DOCU Descriptors:	Many lifecycle needs uncovered	Some lifecycle needs uncovered.	Right-sized to lifecycle needs	Excessive for lifecycle needs	Very excessive for lifecycle needs	
Rating level	Very Low	Low	Nominal	High	Very High	Extra High
Effort multiplier	0.81	0.91	1.00	1.11	1.23	n/a

3.2.2 Platform factors

- **Execution Time Constraint (TIME)**

This factor takes into account the execution time constraint imposed upon a software system. It is calculated as the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource.
We have properly chosen efficient data structures and algorithms so that we don't require much computing power. For this reason the TIME factor would be Nominal.

TIME Descriptors:			≤ 50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
Rating level	Very Low	Low	Nominal	High	Very High	Extra High

Effort multiplier	n/a	n/a	1.00	1.11	1.29	1.63

- **Main Storage Constraint (STOR)**

This rating represents the degree of main storage constraint imposed on a software system or subsystem. Since we do not have data-intensive application, we will require a Nominal rating

- **Platform Volatility (PVOL)**

This factor take into account how often the platform receive updates. For our purpose, the platform is JEE and receives a major update every 1 year or so, so this lead to Low PVOL.

3.2.3 Personnel Factors

- **Analyst Capability (ACAP)**

This factor rates the analyst, namely the personnel who work on requirements, high-level design and detailed design. The rating should be based on their design and communication ability as well as cooperation and efficiency. We believe to belong in the 55th percentile (Nominal ACAP) because we did a complete and detailed analysis of the software system.

- **Programmer Capability (PCAP)**

This factor rates the programmer productivity, efficiency and ability to cooperate in teams. We both believe to belong in the 55th percentile, giving us a Nominal PCAP

- **Personnel Continuity (PCON)**

Since we are only two people in the team, we expect a low continuity because we may want to hire more people. This said, we have chosen this factor to be Low.

- **Applications Experience (APEX)**

We both have no experience with JEE, so this factor is set to Very Low

- **Platform Experience (PLEX)**

We had some experience in databases, user interfaces and networking, so we have chosen this parameter to be Low

- **Language and Tool Experience (LTEX)**

We both know and code in Java for over 1 year, so this gives a Nominal LTEX factor.

3.2.4 Project Factors

- **Use of Software Tools (TOOL)**

This factor takes into account how much the software tools used affect the productivity. Since we will use strong, mature lifecycle and moderately integrated tools we expect to have a High rating.

- **Multisite Development (SITE)**

This factor deals with the collocation of the members of the team during the development. This includes both site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia).
Since we are going to work at home and occasionally meet each other, SITE factor will be High.

- **Required Development Schedule (SCED)**

This rating measures the schedule constraint imposed on the project team developing the software. Since we won't impose any accelerated schedule, we have a Nominal rating.

3.2.5 Overall estimation of cost drivers

The following table shows an overview of the cost drivers defined above.

Cost driver	**Rating level**	**Effort multiplier**
RELY	High	1.10
DATA	High	1.14
CPLX	*Control*: High *Computation*: Nominal *Device-dependant*: Nominal *Data management*: High *UI management*: Nominal	1.102
RUSE	High	1.07
DOCU	Nominal	1.00
TIME	Nominal	1.00
STOR	Nominal	1.00
PVOL	Low	0.87
ACAP	Nominal	1.00
PCAP	Nominal	1.00
PCON	Low	1.12
APEX	Very Low	1.22
PLEX	Low	1.09
LTEX	Nominal	1.00

TOOL	High	0.90
SITE	High	0.93
SCED	Nominal	1.00
Result		1.60

This cost driver analysis gives us the EAF:

$$EAF = \prod_{i=1}^{16} EM_i = 1.60$$

3.3 Effort and schedule estimation

Now that we have all the ingredients, we can plug the scale factors and cost drivers in the effort equation to estimate the project overall cost.
The effort equation, as for COCOMO II, is:

$$Effort = A \times Size^E \times \prod_{i=1}^{n} EM_i = A \times Size^E \times EAF$$

where $E = B + 0.01 \times \sum_{j=1}^{5} SF_j$ and $A = 2.94,\ B = 0.91$

that gives us:

$$E = 0.91 + 0.01 \times \sum_{j=1}^{5} SF_j = 0.91 + 0.01 \times 18.57 = 1.096$$

$$Effort = 2.94 \times 7.146^{1.096} \times 1.60 = 40.6\ PM$$

and an upper bound of

$$Effort_{max} = 2.94 \times 10.452^{1.096} \times 1.60 = 61.6\ PM$$

Concerning the time to develop estimation (TDEV), the formula as for COCOMO II is:

$$TDEV = C \times (PM)^{D + 0.2 \times (E-B)} \times \frac{SCED\%}{100}$$

where $C = 3.67,\ D = 0.28,\ B = 0.91$

that gives us:

$$TDEV = 3.67 \times 40.6^{\,0.28 + 0.2 \times (1.096 - 0.91)} \times 1.0 = 11.88\ months$$

with an upper bound of:

$$TDEV_{max} = 3.67 \times 61.6^{\,0.28 + 0.2 \times (1.096 - 0.91)} \times 1.0 = 13.56\ months$$

4 Tasks and schedule

In order to define deadlines for the development of the system, we have identified a suitable schedule of tasks that will allow us to complete the system with reasonable confidence. We divided the tasks in four groups:

Project planning:

The first part of the project is of course the planning. In this part we define 4 documents to be delivered, and the final presentation to the stakeholders.
The deadlines that we have set for these deliveries are the same that we had at our disposal.
This stage is composed of:

1. Requirement Analysis and Specification Document (RASD) preparation.
2. Design Document (DD) preparation.
3. Integration Test Plan Document (ITPD) preparation.
4. Project Plan (PP) preparation.
5. Final presentation.

Development phase 1:

After retrieving all requisites, we can start the actual coding. This first coding phase is intended to produce a mostly complete prototype that can then be used to validate the requirements defined before.
The main tasks of this stage are:

1. Development of the main classes, with unit tests.
2. Code inspection of the written code
3. Requirement validation, in which we check that the main classes cover the requirements specified in the RASD

Development phase 2:

Development phase 2 is intended to complete the project with all its functionalities. The second half of this phase will mostly be spent in resolving bugs, improving security, code documentation, maintenance and so on. This phase will be mostly overlapped with the testing phase, as we forecast some integration issues that should be resolved by acting on the code itself.
The last part of this stage will be dedicated to code inspection. Here is a brief summary:

1. Development of the remaining classes.
2. Code inspection on the implemented classes.

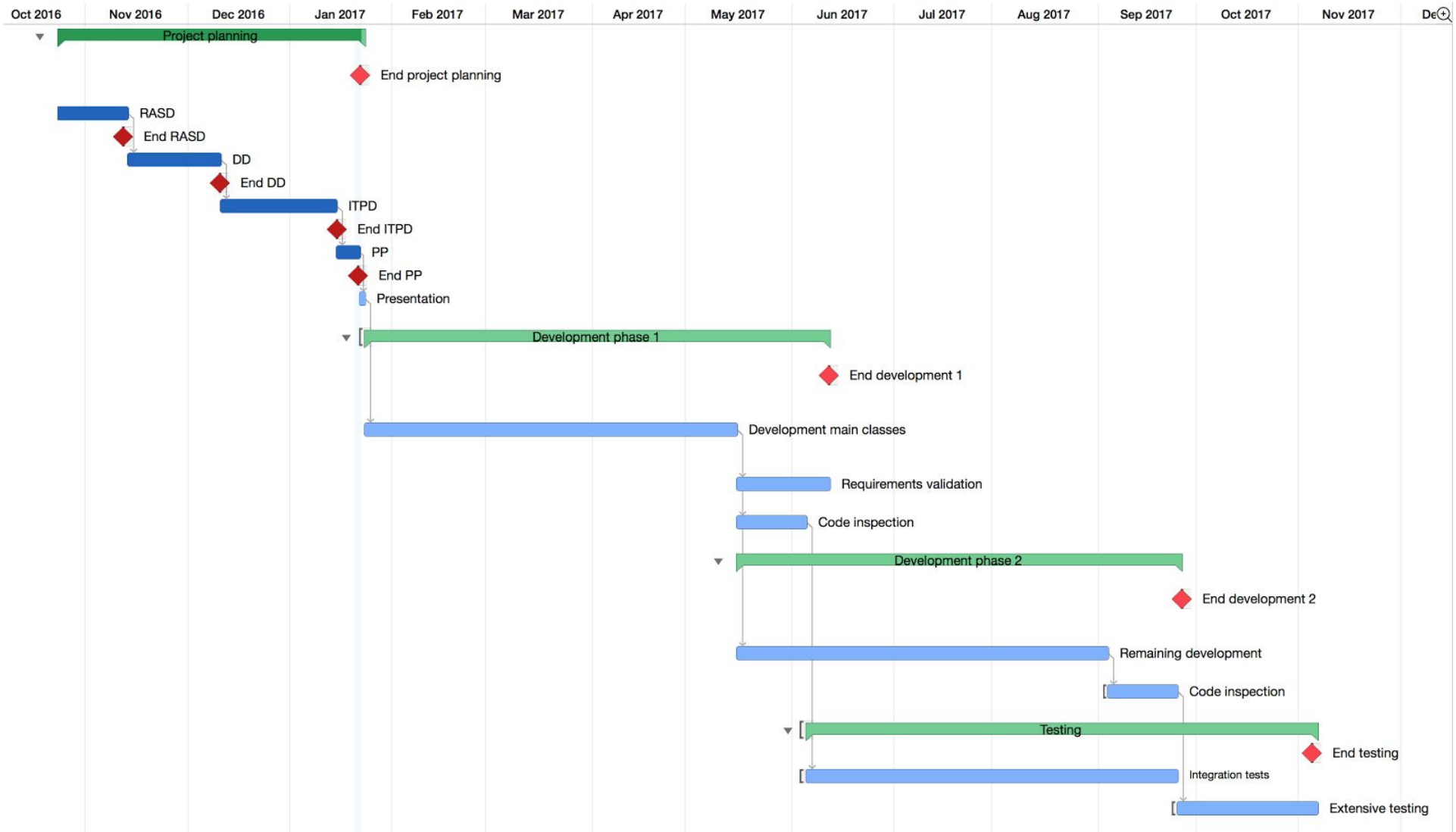
Testing phase:

The testing phase can start only after the code inspection has been completed on the first development phase and lasts 2 weeks after the end of the second development phase. During this phase, two tasks will be carried out:

1. Integration tests (and development of mockup classes).
2. Extensive testing of the system (alpha stage).

This is a summary of the tasks and milestones that we have defined:

Task	Start	End	Duration	Completed	Dependencies
0) PowerEnjoy	**24/10/16 08:00**	**06/11/17 17:00**	**13mo 2w 1d**	**17%**	
1) Project planning	**24/10/16 08:00**	**23/01/17 17:00**	**3mo 1w 1d**	**98%**	
1.1) End project planning	22/01/17 00:00	22/01/17 00:00		0%	
1.2) RASD	24/10/16 08:00	11/11/16 17:00	3w	100%	
1.3) End RASD	11/11/16 08:00	11/11/16 08:00		100%	
1.4) DD	14/11/16 08:00	09/12/16 17:00	1mo	100%	1.2
1.5) End DD	09/12/16 16:00	09/12/16 16:00		100%	
1.6) ITPD	12/12/16 08:00	13/01/17 17:00	1mo 1w	100%	1.4
1.7) End ITPD	13/01/17 16:00	13/01/17 16:00		100%	
1.8) PP	16/01/17 08:00	20/01/17 17:00	1w	100%	1.6
1.9) End PP	20/01/17 13:00	20/01/17 13:00		100%	
1.10) Presentation	23/01/17 08:00	23/01/17 17:00	1d	0%	1.8
2) Development phase 1	**24/01/17 08:00**	**12/06/17 17:00**	**5mo**	**0%**	
2.1) End development 1	12/06/17 16:00	12/06/17 16:00		0%	
2.2) Development main classes	24/01/17 08:00	15/05/17 17:00	4mo	0%	1.10
2.3) Requirements validation	16/05/17 08:00	12/06/17 17:00	1mo	0%	2.2
2.4) Code inspection	16/05/17 08:00	05/06/17 17:00	3w	0%	2.2
3) Development phase 2	**16/05/17 08:00**	**26/09/17 16:00**	**> 4mo 3w 0,75d**	**0%**	
3.1) End development 2	26/09/17 16:00	26/09/17 16:00		0%	
3.2) Remaining development	16/05/17 08:00	04/09/17 17:00	4mo	0%	2.2
3.3) Code inspection	05/09/17 08:00	25/09/17 17:00	3w	0%	3.2
4) Testing	**06/06/17 08:00**	**06/11/17 17:00**	**5mo 2w**	**0%**	
4.1) End testing	03/11/17 16:00	03/11/17 16:00		0%	
4.2) Integration tests	06/06/17 08:00	25/09/17 17:00	4mo	0%	2.4, 2.4
4.3) Extensive testing	26/09/17 08:00	06/11/17 17:00	1mo 2w	0%	3.3


Oct 2016
Nov 2016
Dec 2016
Jan 2017
Feb 2017
Mar 2017
Apr 2017
May 2017
Jun 2017
Jul 2017
Aug 2017
Sep 2017
Oct 2017
Nov 2017
Project planning
End project planning
RASD
End RASD
DD
End DD
ITPD
End ITPD
PP
End PP
Presentation
Development phase 1
End development 1
Development main classes
Requirements validation
Code inspection
Development phase 2
End development 2
Remaining development
Code inspection
Testing
End testing
Integration tests
Extensive testing

5 Resource allocation

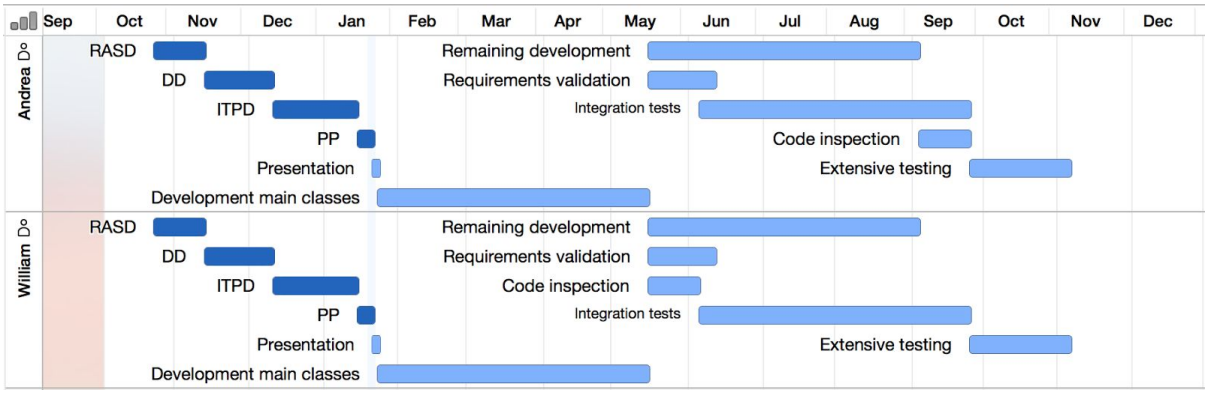
To ensure a smooth development course, we will lay out a detailed resource allocation which will allow us to specify in great detail the tasks that each team member will be in charge of. We will keep into consideration the time available to each team member (e.g. some developer might be busy in some specific week).

To allocate the resources, we will assume a work day of 8 hours/day, keeping in mind that weekends and national holidays have to stay free.

We decided to split evenly the workload for every task specified above. Here is a summary of the effort required for the completion of the project:

Name	Start Date	End Date	Duration	% Complete
Andrea	**24/10/16 08:00**	**06/11/17 17:00**	**18mo 2w 1d**	**18%**
RASD	24/10/16 08:00	11/11/16 17:00	3w	100%
DD	14/11/16 08:00	09/12/16 17:00	1mo	100%
ITPD	12/12/16 08:00	13/01/17 17:00	1mo 1w	100%
PP	16/01/17 08:00	20/01/17 17:00	1w	100%
Presentation	23/01/17 08:00	23/01/17 17:00	1d	0%
Development main classes	24/01/17 08:00	15/05/17 17:00	4mo	0%
Requirements validation	16/05/17 08:00	12/06/17 17:00	1mo	0%
Remaining development	16/05/17 08:00	04/09/17 17:00	4mo	0%
Integration tests	06/06/17 08:00	25/09/17 17:00	4mo	0%
Code inspection	05/09/17 08:00	25/09/17 17:00	3w	0%
Extensive testing	26/09/17 08:00	06/11/17 17:00	1mo 2w	0%
William	**24/10/16 08:00**	**06/11/17 17:00**	**18mo 2w 1d**	**18%**
RASD	24/10/16 08:00	11/11/16 17:00	3w	100%
DD	14/11/16 08:00	09/12/16 17:00	1mo	100%
ITPD	12/12/16 08:00	13/01/17 17:00	1mo 1w	100%
PP	16/01/17 08:00	20/01/17 17:00	1w	100%
Presentation	23/01/17 08:00	23/01/17 17:00	1d	0%
Development main classes	24/01/17 08:00	15/05/17 17:00	4mo	0%
Code inspection	16/05/17 08:00	05/06/17 17:00	3w	0%
Remaining development	16/05/17 08:00	04/09/17 17:00	4mo	0%
Requirements validation	16/05/17 08:00	12/06/17 17:00	1mo	0%
Integration tests	06/06/17 08:00	25/09/17 17:00	4mo	0%
Extensive testing	26/09/17 08:00	06/11/17 17:00	1mo 2w	0%

As you can see, we have split the code inspection in 2 parts and the two parts are assigned to different people. We opted to do this so that to minimize the possibility that the same error could be ignored due to personal attitudes.



6 Risks and recovery actions

Risks are defined as the probability of suffering loss. Note that there is a difference between a problem and risk: a problem is an event that has already occured while risks are generally unpredictable, you can only estimate the probability in which they can happen and their impact on the project.
For each risk we will define the probability of the event (Rare, Unlikely, Possible, Frequent) and the impact (Very Low, Low, Average, High, Very High) it would have if it happened.
From these parameters we can prioritize the risks on the risk exposure, that is the product of the probability and the impact.

6.1 Identified risks

The risks are subdivided in three categories: project risks, technical risks and business risks.

6.1.1 Project risks

The project risks harm the project plan and usually increase the development time more than expected, thus increasing the cost.
We have identified the following project risks:

Risk identifier	**Risk description**	**Probability**	**Impact**
Req1	Analysts misunderstood the requirements	Possible	Very High
Req2	Stakeholders change the requirements during development phase	Possible	High
Req3	Analysts underestimate the complexity of the system	Rare	Very High
Pers1	Developers are not able to develop a complex JEE system	Possible	Very High
Pers2	Team manager is absent during critical development phases	Unlikely	High
Pers3	Some team members are absent during critical development phases	Unlikely	High
Pers4	Miscommunication between team members	Possible	Average

6.1.2 Technical risks

Technical risks threaten the quality and timeliness of the software to be produced. If they become real, implementation may become difficult or impossible
We have identified the following technical risks:

Risk identifier	Risk description	Probability	Impact
Design1	The architecture identified in the Design Document is not suitable for the kind of software to be developed	Unlikely	Very High
Design2	The algorithms identified in the design document turn out to be wrong, hard to implement or even unfeasible	Unlikely	High
Impl1	The client is not satisfied by the overall system performance	Likely	Average
Impl2	Code is not well documented and difficult to debug / test / maintain. Code contains bugs that are difficult to find and/or fix.	Likely	Average
Test1	Testing requires more stubs or drivers than expected	Unlikely	High

6.1.3 Business risks

Business risks threaten the financial aspect of the software to be produced; if one of them becomes real it can compromise the economic success of the project.
We have identified the following business risks:

Risk identifier	Risk description	Probability	Impact
Market1	There is no demand in the product. The service is used by a very limited number of people	Possible	Very High
Market2	Other competitors develop a similar service in the same area.	Unlikely	High
Budget1	The budget to develop this project is reduced due to financial problems or any other problem.	Unlikely	High

6.2 Risk strategy

In the next table we will describe how to possibly avoid the risks defined above and what to do in case they become real.

Risk	Prevention	Correction
Req1	- Adopt a formalized method for requirement engineering - Perform requirements validation once a month	Recycle on requirement retrieval and analysis. Fix the RASD.
Req2	- Explain to client that once the RASD is accepted it should not be modified until the end of development phase - Make sure that the stakeholders are pretty confident with the established requirements.	Recycle on requirement retrieval and analysis. Fix the RASD.
Req3	- Make sure the analyst is competent and has experience with the kind of system to be developed - Ask external experienced analysts for a consultation	Recycle on requirement retrieval and analysis. Fix the RASD.
Pers1	- Provide courses on JEE.	Hire some experienced developer that knows JEE.
Pers2	- Plan the meetings in advance so that to minimize this risk - Nominate a vice manager	Vice manager takes the role of the manager
Pers3	- Team members should warn in advance of their absence - Make sure the knowledge is shared among team members	Another team member takes care of the work of the absent one.
Pers4	- Always make a report after every meeting - Encourage discussion on the choices to be made.	Try to resolve the conflict in a meeting with the involved group members.
Design1	- Perform inspection of the DD. - Ask an advisor about the feasibility of the DD.	Cycle on design process and update the DD.
Design2	- Perform inspection of the DD. - Use a simulation tool to state the	Revise the algorithm and change it.

	correctness of the algorithm	
Impl1	- Make sure that performance is taken into account in requirement analysis and during system design	- Revise the algorithms used to see if a better algorithm could be used. - Revise the architecture and design choices in order to reach the required performance
Impl2	- Always put comments in the code - Adopt pair programming technique.	Apply code inspection on the code
Test1	- Adopt inspection on ITDP.	Implement the missing stubs and drivers
Market1	- Provide a good user interface design - Make the application easy to use	- Advertise the product - Do market research on possible customers and tailor the application for them.
Market2	- Make the application	- Advertise the product - Differentiate from the competitors by offering new functionalities
Budget1	- Make sure that there is the availability of the budget at the beginning of the development	- Show to the stakeholders a working prototype and try to convince them about the effectiveness of the product.

7 Appendix

7.1 Used tools

- Google Docs (http://www.docs.google.com) for this document
- Star UML (http://staruml.io/) for the UML modelling, in particular use case, statechart, activity, class and sequence diagrams
- Alloy Model 4.2 (http://alloy.mit.edu/alloy) to generate the world and proving its consistency
- Balsamiq Mockup (http://balsamiq.com/products/mockups/) for the sketch on the user interfaces of this software system
- Omni Plan (https://www.omnigroup.com/omniplan) for the Gantt diagrams and resource allocation.

7.2 Hours of work

The following are the total number of hours spent by each member of the group:

- Andrea Battistello: 15 h
- William di Luigi: 10 h

7.3 Revision history

Version	*Date*	*Revision description*	*Revision notes*
1.0	22-01-2017	First release	-