POLITECNICO
MILANO 1863

Politecnico di Milano
A.A. 2016–2017
Software Engineering 2: "PowerEnJoy"

Project Plan

Pietro Ferretti, Nicole Gervasoni, Danilo Labanca

January 21, 2017

Contents

1 Introduction **3**
- 1.1 Purpose . . . 3
- 1.2 Scope . . . 3
- 1.3 List of Definitions and Abbreviations . . . 4
  - 1.3.1 Definitions . . . 4
  - 1.3.2 Acronyms . . . 4
- 1.4 List of Reference Documents . . . 4

2 Project size, cost and effort estimation **5**
- 2.1 Size estimation: function points . . . 6
  - 2.1.1 Internal Logic Files (ILFs) . . . 6
  - 2.1.2 External Logic Files (ELFs) . . . 7
  - 2.1.3 External Inputs (EIs) . . . 8
  - 2.1.4 External Inquiries (EQs) . . . 10
  - 2.1.5 External Outputs (EOs) . . . 11
  - 2.1.6 Overall estimation . . . 12
- 2.2 Cost and effort estimation: COCOMO II . . . 12
  - 2.2.1 Scale Factors . . . 12
  - 2.2.2 Cost Drivers . . . 13
  - 2.2.3 Effort equation . . . 17
  - 2.2.4 Schedule estimation . . . 18

3 Schedule **18**

4 Resource allocation **18**

5 Risk Management **18**

A Changelog **20**

B Hours of work **20**

1 Introduction

1.1 Purpose

The purpose of this document is to provide a detailed analysis of the PowerEnjoy software development project in terms of required cost and time. It highlights the estimation of

- project size, calculated using the *Function Points approach* by IBM;
- project cost and effort, calculated using the *COCOMO II* by Boehm.

Given the previous information we elaborate a feasible schedule considering all the necessary activities in detail, thus the best resources' allocation on each one. The last section of the document focuses on handling all the possible risks that could be met during the whole process, from the requirements analysis to the final testing and deployment.

1.2 Scope

The aim of this project is to specify and design a new digital management software for PowerEnJoy, a car-sharing service that employs electric cars only.

PowerEnJoy will offer a very valuable service to its users, letting them borrow cars to drive around the city freely, as an alternative to their own vehicles and public transport. Among the advantages of using PowerEnJoy we can note being able to find available cars in any place that is served by our system and having dedicated spots to park in (namely, PowerEnJoy's power grid stations). Furthermore, thanks to the fact that all the cars that we provide are electrically powered, PowerEnJoy is also very environmentally friendly.

1.3 List of Definitions and Abbreviations

1.3.1 Definitions

1.3.2 Acronyms

- **ITPD**: Integration Test Plan Document
- **DD**: Design Document
- **RASD**: Requirements Analysis and Specification Document
- **DB**: Database
- **PGS**: Power Grid Station
- **GPS**: Global Positioning System
- **API**: Application Programming Interface
- **ISDTN**: International Standard Date and Time Notation
- **EM**: Effort Multiplier
- **FP**: Function Points
- **ILF**: Internal Logic File
- **ELF**: External Logic File
- **EI**: External Input
- **EO**: External Output
- **EQ**: External Inquiries
- **UI**: User Interface

1.4 List of Reference Documents

- Requirements analysis and specification document: "RASD.pdf"
- Design document: "DD.pdf"
- Integration testing document: "ITPD.pdf"
- Project description document: "Assignments AA 2016-2017.pdf"
- Example document: "Project planning example document.pdf"
- "COCOMO II – Model Definition Manual", version 2.1, 1995-2000, Center for Software Engineering, USC

2 Project size, cost and effort estimation

Pay a Bill	
Input	*Result*
A valid session token, a bill that needs to be paid and a valid payment method	The transaction is carried out; if it succeeds the bill is marked as paid, otherwise returns failure.
A valid session token, a bill that needs to be paid and an ill-formed payment method	An exception is raised.
A valid session token and a bill that needs to be paid	The system uses the payment method saved for the user to carry out the transaction; if it succeeds the bill is marked as paid, otherwise returns failure.
A valid session token and a bill that has already been paid	An exception is raised.
A valid session token and a non-existent bill	An exception is raised.
An invalid session token and a bill	An exception is raised (bad authentication).

ELF	**Complexity**	**FPs**
elf n1	Low	5
elf n2	High	10
elf n3	medium	7
Total		**22**

Table 1: asdfasdf

ELF	**Complexity**	**FPs**
elf n1	Low	5
elf n2	High	10
elf n3	medium	7
	total	**22**

Table 2: ewerewr

Cost Driver	Rating Level	EM
Documentation match to life-cycle needs (DOCU)	Nominal	1.00
Total		**1.00**

Table 3: I'm a table. Are u sure?

2.1 Size estimation: function points

Function points are useful in expressing the amount of business functionality our software has to provide to a user and are used to compute an estimation of its size. After been identified and categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces, each functional requirement is then assessed for complexity and assigned a number of function points. We based our computation on tables and values in *COCOMO II Model Definition Manual v. 2.1.*

2.1.1 Internal Logic Files (ILFs)

They are all kinds of data used and managed by the application in order to offer the expected functions.
Data will be organize in the following tables in the DB:

- **User** : name, surname, username, password, dob, email, licenseID, cvv, cardNumber, accountStatus
- **Bill** : associatedLicense, total, date, rideID, carID, paymentStatus
- **Car** : model, plate, ID, available, issues
- **Report** :carID, description, associatedLicense, date
- **Safe area** : latitude, longitude, ID
- **PGS** : latitude, longitude, ID
- **Plug** : available, ID
- **Reservation** : ID, associatedLicense, carID, date, status
- **Ride** : ID, associatedLicense, associatedBill, date, status, ridingTime, carID

The software will operate directly on the previously listed data and with the tables generated from their relations between each other.
All this data are modeled in simple structures so their complexity can be considered low (referring to tables).

ILF	Complexity	FPs
User	Low	7
Bill	Low	7
Car	Low	7
Report	Low	7
Safe area	Low	7
PGS	Low	7
Plug	Low	7
Reservation	Low	7
Ride	Low	7
Total		73

quel +10 cos'è?

$$FPs(ILF) = 7 \times 9 + 10 = 73$$

2.1.2 External Logic Files (ELFs)

The situations in which our system demands external data is when it needs informations regarding geolocation or when it must guarantee the legal soundness of the Driving License.
In particularly:

- **GraphHopper API**:
  - Given the string containing the address, the API returns a pair of float representing the coordinates of that location.
  - Given two pair of coordinates, the API return a float representing the time within two position.
- **Eucaris API**:
  - Given name, surname, driving license ID and expiration date as string, the API returns a boolean value representing the correspondence with an existing driving license in Eucaris DB.

In the final analysis, as the involved data are string and number with restrained size, we can assess this logic files as low complexity.

ELF	Complexity	FPs
Reverse geocoding	Low	5
Isochrone distance	Low	5
Driving Licenes legal soudness	Low	5
Total		15

2.1.3 External Inputs (EIs)

PowerEnjoy offers a remarkable series of functionalities that required user's input.
In particularly:

- **Login**: this functionality demands only two strings as parameters, the username and the password, that will be compared with the ones stored in the DB. We can consider as a low complexity operation.
- **User update**: this functionality includes a collection of operations that allow to modify each aspect of user's profile. The input data are simple strings. Since the possibility are conspicuous and the different elaborations aren't basic and futhermore they interest several components, we can regard this functionality as an average complexity operation.
- **Pay bill** (automatically/manually): this is one of the most complex operations. It involves internal components and external APIs and it demands two numbers as input. Given the relevance of the operation and the parts interested, we can consider this as high complexity operation.
- **Create reservation**: this functionality requires as input the user ID and car ID, that we can consider simple inputs, while his complexity is due to the components involved. In fact at the initial moment of the creation, the application verifies if the user is suspended and after that is modified the availabilty field of the tuple representing the car and the car itself is put in order to be unlocked. Moreover it's inserted in the list of reservation made by the user this new one. Given that we classify the operation as having average complexity.
- **Cancel reservation**: the analysis made for the previous functionality is valid also for this one. The operations are comparable as elaboration and thanks to this the complexity of this functionality is average.
- **Start ride**: this action is very simple as it's demands only the user ID and the car ID and it tallies the duration of the ride. Made this consideration we assues that this functionality as low complexity.

- **End ride**: this functionality is the most complex one because involves various components and includes the payment functionality that has high complexity. When a ride ends the car is set as available in the DB, it creates a bill for the user and initiates the payment. As usual the inputs are simple strings.
- **Park**: this functionality expects as inputs the position of the car that are a pair of numbers, the user ID and the car ID. In the elaboration of this data the components interested are the internal car system and the DB to verify if the position belongs to a safe area. Due to the urgency of the action this functionality has an average complexity.
- **Unlock car**: this functionality receives the position of the user and his ID and the car ID. After that the system proceeds to notify the car to open the doors and calls the *start ride* function. Since it comprises another functionality and communicates with the car we classify this functionality with average complexity.
- **Update car**: this functionality requires only two parameters: the car ID and the status to be update. The component involved is the DB. For these reasons this functionality has low complexity.
- **Update plug**: as the previous functionality, also this one requires two parameters and affects the DB. So it's a low complexity functionality.
- **Set car available**: as the previous functionality, also this one requires two parameters and affects the DB. So it's a low complexity functionality.
- **Set car available**: as the previous functionality, also this one requires two parameters and affects the DB. So it's a low complexity functionality.
- **Report issue**: this functionality receives a form as input, so a set of strings, and modifies the status of the car in the DB and car itself. So we can consider this functionality as average complexity operation.

EI	Complexity	FPs
Login	Low	3
User update	Average	4
Pay bill	High	6x2
Create reservation	Average	4
Cancel reservation	Average	4
Start ride	Low	3
End ride	High	6
Park	Average	4
Unlock car	Average	4
Update car	Low	3
Update plugs	Low	3
Set car unavailable	Low	3
Set car available	Low	3
Report issue	Average	4
Total		60

2.1.4 External Inquiries (EQs)

In this section we will discuss about External Inquiries that could be defined as elementary processes that send data or control information outside the application boundary.; the primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF of EIF. (copiata)
In our case we have:

- **Get user info**: it's returned a set of strings representing the user's info, from the DB.
- **Get bills**: it's returned a list of bills, made by strings, of the user, from the DB.
- **Car/PGS/Safe area search with position**: in this case the function needs a parameter that is a pair of number and returns a list extracted from the DB. For these operations there are some elaborations to do and because of that we classify these actions having average complexity.
- **Car/PGS/Safe area search with address**: as for the previous cases, also these functionalities have an average complexity even though there is

an interaction with the *GraphHopper API* in order to convert the address in coordinates.

- **Money Saving Option**: this is the most complex operation because it should extract information from the DB and elaborate them to find the best solution.
- **Cars in need of maintenance**: this is a simple functionality that is employed by maintenance operators and returns a list of cars that are unavailable.

EQ	Complexity	FPs
Get user info	Low	3
Get bills	Low	3
Car/PGS/Safe area search with position	Average	4x3
Car/PGS/Safe area search with address	Average	4x3
Money Saving Option	High	6
Cars in need of maintenance	Low	3
Total		39

2.1.5 External Outputs (EOs)

The situation our system needs to notify external agents are the following:

- **Un/Lock car**
- **Car update**

The first case has a low complexity while the second is a little bit more complex because it needs to extract the information from the car.

EO	Complexity	FPs
Un/Lock car	Low	4x2
Car update	Average	5
Total		13

2.1.6 Overall estimation

Function Type	FPs
Internal Logic Files	73
External Logic Files	15
External Inputs	60
External Inquiries	39
External Outputs	13
Total	200

This table contains a recap of the evaluations of each type of function point. Thanks to estimation of the function points we can appraisal the approximately number of lines of code: since that our system is developed using Java EE, we can extrapolate the AVC factor from this table[1].

$$SLOC = AVC \times numberoffunctionpoints$$

where
AVC is equale to 46 $numberoffunctionpoints$ is equal to 200
So at the end, we have:

$$SLOC = 9200$$

2.2 Cost and effort estimation: COCOMO II

We use the COCOMO II Model for cost and effort estimation. *cos'è il COCOMO II?*

We use the Post-Architecture Model, because the architecture and everything has already been described in depth.

qualche dettaglio

2.2.1 Scale Factors

what are scale factors

qua ci vuole la tabella dove ci sono tutti i valori
Scale factors:

- **Precedentedness (PRED):** "If a product is similar to several previously developed projects, then the precedentedness is high." how similar is this product to previously developed one how many similar product have we developed already We never developed anything similar [and on this scale], except for the client-server architecture. We rate this factor as Very Low.

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

- **Development Flexibility (FLEX):** "Need for software conformance with pre-established requirements & external interface specifications, + premium on early completion" We have to 'considerably' follow the requirements we specified to reach the customer's goals. We rate this as Nominal.
- **Architecture / Risk Resolution (RESL):** High if everything is well planned, the risks have all been identified and accounted for, the architecture has been carefully designed, low uncertainty and low risk in general. our architecture is good our risk management plan is (we consider it) thorough We rate this as High.
- **Team Cohesion (TEAM):** experience as a team, willingness to work together, shared vision and commitments. We are great at this, we rate this as Very High.
- **Process Maturity (PMAT):** follows the Capability Maturity Model (CMM). nb: level 1 -> inital, uncontrolled level 2 -> managed level 3 -> defined level 4 -> quantitatively managed level 5 -> optimized we are not at a level where everything is accounted for, but we do follow a clear process Around level 3, equivalent to a High.

Scale Factor	Level	Value
Precedentedness (PREC)	Very Low	6.20
Development Flexibility (FLEX)	Nominal	3.04
Architecture / Risk Resolution (RESL)	High	2.83
Team Cohesion (TEAM)	Very High	1.10
Process Maturity (PMAT)	High	3.12
Total		16.29

2.2.2 Cost Drivers

what are cost drivers each of these factors can increase or decrease the amount of effort needed to develop the software

We follow the tables in the COCOMO II document.

Cost Drivers:

- **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.

  Even if our system doesn't work the customers have alternatives. And we implement safeguards in the cars not to get stuck inside or anything

else. Our software doesn't have any way to control the way the cars drive. Moderate, easily recoverable losses.

All in all, *Nominal*, 1.00

- **Database Size (DATA):** This cost driver attempts to capture the effect large test data requirements have on product development. The rating is determined by calculating D/P, the ratio of bytes in the testing database to SLOC in the program.

  We *estimate* our testing database to be around 1GB of size. Given that we estimated the source code to be around 9000 lines long, this brings the D/P ratio to a order of magnitude of 10^5.

  Very High, 1.28

- **Product Complexity (CPLX):** Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations.

  Checking on the COCOMO tables for product complexity and averaging the results we judge the complexity rating to be Nominal on average.

  Nominal, 1.00

- **Developed for Reusability (RUSE):** This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects.

  We have no requirements asking us to develop code ready to be reused in other projects or products. No reusability constraints.

  Low, 0.95

- **Documentation Match to Life-Cycle Needs (DOCU):** In COCOMO II, the rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs.

  We have no specific instructions about documentation, so we're going standard; documentation just right for the product lye-cycle needs.

  Nominal, 1.00

- **Execution Time Constraint (TIME):** This is a measure of the execution time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource.

  Our software will be heavy on performance? Or more like "hardware is expensive, let's keep it active at 70% all the time on average". No waste of processing time.

  High, 1.11

- **Main Storage Constraint (STOR):** This rating represents the degree of main storage constraint imposed on a software system or subsystem.

  Storage is really, really cheap today. We're not in 2000 anymore. We can buy 1000TB of hard disk and be on the safe side easy.

  Nominal, 1.00

- **Platform Volatility (PVOL):** "Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. This rating ranges from low, where there is a major change every 12 months, to very high, where there is a major change every two weeks.

  We want our software to be kept updated to fix bugs and possible vulnerabilities. A major update every 2 months and minor ones every week seems reasonable.

  High, 1.15

- **Analyst Capability (ACAP):** Analysts are personnel who work on requirements, high-level design and detailed design. The major attributes that should be considered in this rating are analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate.

  We are average analysts, nothing special as abilities.

  Nominal, 1.00

- **Programmer Capability (PCAP):** Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate. also capability not experience

  We are very confident in our abilities in programming, great talents. And, we can work as a team

  Very High, 0.76

- **Personnel Continuity (PCON):** The rating scale for PCON is in terms of the project's annual personnel turnover: from 3%, very high continuity, to 48%, very low continuity

  (Not sure if this is correct) We are going to be the ones developing this piece of software, and we are pretty dedicated to following this through until the end. We rate the probability of any of us leaving as negligible.

  Very High, 0.81

- **Applications Experience (APEX):** The rating for this cost driver is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application.

We don't have much experience with this type of application, except for some basic client-server architecture.

Low, 1.10

- **Platform Experience (PLEX):** Understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.

  We have close to zero experience with database, networking, and distributed middleware software.

  Very Low, 1.19

- **Language and Tool Experience (LTEX):** This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem.

  We have experience with Java, and a little JEE too. Good enough, rating this as Nominal (not high, not low).

  Nominal, 1.00

- **Use of Software Tools (TOOL):** What are the capabilities of the tools and the editors we are going to use.

  Modern Java IDEs like Eclipse and Netbeans can do almost anything: basically "strong, mature, proactive life-cycle tools, well integrated with processes, methods, reuse". git integration, sonar, hints ecc.

  Very High, 0.78

- **Multisite Development (SITE):** averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia)

  "The developers"/We are located more or less in the same metro area, and are able to meet whenever to solve problems. Also communication is not a problem (whatsapp, ecc.) + Trello (tools for organization and issue distribution non so)

  High, 0.93

- **Required Development Schedule (SCED):** This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort.

  We have no constraint on time or schedules. We can just keep the spontaneous time required for the development, with no schedule acceleration or stretching. (100%)

  Nominal, 1.00

Cost Driver	Level	Value
Required Software Reliability (RELY)	Nominal	1.00
Database Size (DATA)	Very High	1.28
Product Complexity (CPLX)	Nominal	1.00
Required Reusability (RUSE)	Low	0.95
Documentation match to lyfe-cycle needs (DOCU)	Nominal	1.00
Execution Time Constraint (TIME)	High	1.11
Main Storage Constraint (STOR)	Nominal	1.00
Platform Volatility (PVOL)	High	1.15
Analyst Capability (ACAP)	Nominal	1.00
Programmer Capability (PCAP)	Very High	0.76
Personnel Continuity (PCON)	Very Low	0.81
Application Experience(APEX)	Low	1.10
Platform Experience (PLEX)	Very Low	1.19
Language and Tool Experience (LTEX)	Nominal	1.00
Usage of Software Tools (TOOL)	Very High	0.78
Multisite Development (SITE)	High	0.93
Required Development Schedule (SCED)	Nominal	1.00
Total		0.907

2.2.3 Effort equation

We use the formula for the Post-Architecture Model

PM is the estimated Person-Month effort needed to develop the code.

$$PM = A \times Size^E \times \prod_{i=1}^{17} EM_i$$

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

where

$A = 2.94$ is a calibrated constant

$B = 0.91$ is a calibrated constant

EM_i are the effort multipliers (cost drivers)

SF_j are the scale factors

$Size$ is the estimated thousands of lines of source code (KSLOC)

scale drivers = 6.20 + 3.04 + 1.41 + 1.10 + 3.12 = 14.87 cost drivers (effort multipliers) = 1.0*1.28*1.00*0.95*1.00*1.11*1.00*1.15*1.00*0.76*0.81*1.10*1.19*1.00*0.78*0.93*1.00

$= 0.907$
then
$E = 0.91 + 0.01 \times 16.29 = 1.07$
$PM = 2.94 \times 8.694^{1.06} \times 0.907 = 26.97$ person-months

2.2.4 Schedule estimation

$TDEV$ is the time necessary to develop the code, in calendar months assuming no schedule compression, no stretching-out

$$TDEV = C \times PM^F$$

$$F = D + 0.2 \times (E - B)$$

where
$B = 0.91$ is a calibrated constant
$C = 3.67$ is a calibrated constant
$D = 0.28$ is a calibrated constant
E is the scaling exponent for the effort equation
PM person-months obtained in the effort estimation
then
$F = 0.28 + 0.2 \times (1.07 - 0.91) = 0.31$
$TDEV = 3.67 \times 26.97^{0.31} = 10.19$ months

3 Schedule

4 Resource allocation

5 Risk Management

Risk management is the identification, assessment, and prioritization of risks followed by coordinated and economical application of resources to minimize, monitor, and control the probability and/or impact of unfortunate events.
In this section we list PowerEnjoy project's main risks and outline possible strategies to handle the most critical ones.

Risk	Probability	Impact
Personnel shortfall (recruitment issues, employee illness or accidents, ...)	moderate	catastrophic
Inaccurate Requirements	moderate	critical
Unrealistic Schedule	moderate	critical
Unrealistic Budget	moderate	critical
Stakeholder commitment loss	low	critical
New car rental laws	low	catastrophic
Inability to obtain proper permits from authorities	low	catastrophic
Inability to obtain deal with mobile data provider	low	critical
Issues with hardware supplier (wrong/defected items, late deliveries, ...)	moderate	critical
Wrong user interface	moderate	critical

Possible Solutions

- *Personnel shortfall:* reorganize team so that there is more overlap of work and no single person is fundamental to the project; in case of issues in finding high qualified worker consider buying already developed products and simply adapting them.
- *Issue with hardware supplies:* use and test hardware components as soon as they are available; replace potentially defective components with bought-in components of known reliability.
- *Inaccurate requirements:* derive traceability information to assess requirements change impact; divide software functions in several different module so that modify a requirements will not have effect on all the produced code.
- *Stakeholder commitment loss:* prepare a briefing document showing how the project is making a very important contribution to the goals of the business.
- *Unrealistic budget:* have it reconsidered by showing the stakeholder how the project is making a very important contribution to the goals of the business; buy already developed products and assembly them.
- *Unrealistic schedule:* early in the development, divide bigger activities in smaller ones and hired new people to work on few specific functions then combine them when ready, while later consider buying already developed products and adapting them.

A Changelog

B Hours of work