Politecnico di Milano

A.A. 2015-2016

Software Engineering 2 Project – “SmartCityAdvisor”

Prof. Di NittoElisabetta

Project **P**lan **D**ocument

Mirko Basilico, Federico Dazzan

Table of Contents

1. INTRODUCTION

1.1 Revision History

Version	Date	Authors	Summary
1.0	10-07-2016	Mirko Basilico, Federico Dazzan	First Release (Delivery)

1.2 Purpose and Scope

This document contains the project plan for the smartCityAdvisor system. The project team is composed by two people, Mirko Basilico and Federico Dazzan. Function Points and COCOMO II will be applied to estimate the project size, effort and cost. The project, then, will be divided in tasks and a schedule will be planned for them. The members of the team will be allocated to the tasks taking into account the actual availability for the project. Eventually, project risks will be defined, assessed by relevance and associated to proper recovery actions.

1.3 List of Definitions and Abbreviations

- **RASD:** Requirements Analysis and Specification Document
- **DD:** Design Document
- **ITPD:** Integration Test Plan Document
- **PPD:** Project Planning Document
- **API:** Application Programming Interface

1.4 List of Reference Documents

This document directly refers to the following documents:

- smartCityAdvisor – **R**equirement **A**nalysis and **S**pecification **D**ocument; **D**esign **D**ocument; **I**ntegration **T**est **P**lan **D**ocument

2. FUNCTION POINTS ANALYSIS

The Function Points (FP) is a technique to assess the effort needed to design and develop custom software applications.

The estimation is based on a combination of program characteristics like:

- **Internal Logic File**: homogeneous set of data used and managed by the application
- **External Interface File**: 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 logic files
- **External Inquiry**: elementary operation that involves input and output without significant elaboration of data from logic files.

A weight is associated with each of these FP counts. The total is computed by multiplying each raw count by the weight and summing all partial values:

$$UFC = \sum(\text{number of elements of given type}) \times (\text{weight})$$

The following table shows the coefficients to be used in the UFP (Unadjusted Function Points) computations:

Function Type	Complexity		
	Simple	Medium	Complex
Internal Logic File	7	10	15
External Interface File	5	7	10
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6

INTERNAL LOGIC FILES (ILF)

The system stores the information about:

- **Users (Citizens)**
- **Parking Information**
- **Hospital Information**
- **Transport Station Information**

Each of these entities has a simple structure as it is composed of a small number of fields. Thus, we can decide to adopt for all the 4 the SIMPLE weight.

ILF Function Points = 4*7 = 28

EXTERNAL INTERFACE FILES (EIF)

Parking Availability: this entity could have a structure of simple complexity. So, we can decide to adopt the SIMPLE weight.

Hospital Queue Situation: this entity could have a structure of simple complexity. So, we can decide to adopt the SIMPLE weight.

Public Transport Schedules: this entity could have a structure of average complexity because it is composed of many fields and rows. So, we can decide to adopt the MEDIUM weight.

Get Coordinates from Address: this entity could have a structure of simple complexity. So, we can decide to adopt the SIMPLE weight.

Get Travel Time from coordinates: this entity could have a structure of simple complexity. So, we can decide to adopt the SIMPLE weight.

*ELF Function Points = 3*5 +1*7 = 22*

EXTERNAL INPUTS (EI)

The application interacts with the customer to allow him/her to:

- **User Login/Logout:** these are simple operations, so we can adopt the SIMPLE weight for them.
- **User Registration:** this is a simple operation, so we can adopt the SIMPLE weight for it.
- **Traffic Limitation Request:** this operation involves two entities, the admin and the user. It can still be considered simple, so, again we adopt the SIMPLE weight.

*EI Function Points = 7*4 = 28*

EXTERNAL OUTPUT (EO)

- **Notification to the user:** this is a simple operation, so we can adopt the SIMPLE weight for it.
- **Traffic Limitation:** this is a simple operation, so we can adopt the SIMPLE weight for it.

*EO Function Points = 2*4 = 8*

EXTERNAL INQUIRY (EI)

The application allows customers to request information about:

- **Parking Availability:** this is an operation of high complexity, due to multiple APIs invocations (DCC and TCC), so we can adopt the COMPLEX weight for it.
- **Emergency Room Availability:** this is an operation of high complexity, due to multiple APIs invocations (DCC and TCC), so we can adopt the COMPLEX weight for it.
- **Public Transport Times:** this is an operation of high complexity, due to multiple APIs invocations (DCC and TCC), so we can adopt the COMPLEX weight for it.

*EI FunctionPoints3*6=18*

TOTAL FUNCTION POINTS (UFP) = 28+22+28+8+18 = 104

Assuming that the programming language used for our project will be Java EE, the conversion factor between the total Function Point counts (UFP) and the SLOC is 46. Hence, the SLOC number is estimated as:

SLOC =104 * 46 = 4784

3. COCOMO II ANALYSIS

3.1 SCALE DRIVERS

Scale Drivers are the parameters that reflect the non-linearity of the effort with relation to the number of SLOC. They show up at the exponent of the Effort Equation (Equation 3.1).

Precedentedness: 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. The precedentedness is Low because we have some experience of software design but most of the notions used in this project are new to us.

Development flexibility: reflects the degree of flexibility in the develop-ment process. Very low means a prescribed process is used; Extra high means that the client only sets general goals. We set it to Nominal because we have to follow a prescribed process, but we had a certain degree of flexibility in the definition of the requirements and in the design process.

Risk resolution: reflects the extent of risk analysis carried out. Very low means little analysis, Extra high means a complete a thorough risk analysis. We set it to High, because a rather detailed risk analysis is carried out in chapter 6.

Team cohesion: reflects how well the development team know each other and work together. Very low means very diffcult interactions, Very high means an integrated and effective team with no communication problems. We set it to very high, since the cohesion among the two of us is optimal.

Process maturity: reflects the process maturity of the organization. We set it at Nominal, which corresponds to CMM Level 2: “It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.”

The estimated scale drivers for our project, together with the formula to compute the exponent E, are shown in Table 3.1.

3.2 COST DRIVERS

Cost Drivers are the parameters of the Effort Equation that reflect some characteristics of the developing process and act as multiplicators on the effort needed to build the project. They appear as factors in the Effort Equation (Equation 3.1). Cost Drivers are described in detail in the COCOMO Manual [6, p. 25].

The estimated cost drivers for our project, together with the formula to compute the EAF (Effort Adjustment Factor), are shown in Table 3.2.

3.3 EFFORT

$$\text{Effort} = A * EAF * KSLOC^{E} \tag{3.1}$$

Where:

$EAF = \pi_i C_i$: product of all cost drivers

$E = 0.91 + 0.01 * \sum_i SF_i$: depends on the scale drivers.

$A = 2.94$

KSLOC: Kilo-Source Lines of Code (Estimated in the previous chapter).

The total effort results in **13.7 person-months.**

Code	Name	Factor	Value
PREC	Precedentedness	Low	4.96
FLEX	Development flexibility	Nominal	3.04
RESL	Risk resolution	High	2.83
TEAM	Team cohesion	Very High	1.10
PMAT	Process maturity	Nominal	4.68
Total	$E = 0.91 + 0.01 \times \sum_i SF_i$		1.0761

Table 3.1: Scale Drivers for our project.

Code	Name	Factor	Value
RELY	Required Software Reliability	Nominal	1.00
DATA	Data base size	Nominal	1.00
CPLX	Product Complexity	Nominal	1.00
RUSE	Required Reusability	High	1.07
DOCU	Documentation match to life-cycle needs	Nominal	1.00
TIME	Execution Time Constraint	Nominal	1.00
STOR	Main Storage Constraint	Nominal	1.00
PVOL	Platform Volatility	Low	0.87
ACAP	Analyst Capability	Nominal	1.00
PCAP	Programmer Capability	Nominal	1.00
APEX	Application Experience	Nominal	1.00
PLEX	Platform Experience	Nominal	1.00
LTEX	Language and Tool Experience	Nominal	1.00
PCON	Personnel Continuity	Nominal	1.00
TOOL	Usage of Software Tools	Nominal	1.00
SITE	Multisite Development	High	0.93
SCED	Required Development Schedule	Nominal	1.00
Total	$EAF = \pi_i C_i$		0.866

Table 3.2: Cost Drivers for our project.

3.4 DURATION

$$\text{Duration} = 3.67 * (PM)^{0.28+0.2 * (E - 0.91)} \qquad (3.2)$$

Where:

PM is the effort as calculated in Equation 3.1.

E is the exponent depending on the scale drivers (the same one of the effort equation).

The duration calculated from Equation 3.2 results in a total of **8.7 months.** This would result in 1.6 developers needed for this project.
Since our group consists of 2 people, a reasonable development time would be:

$$\frac{13.7 \text{ pm}}{2 \text{ people}} = 6.8 \text{ months}$$

In order to be cautious with the task scheduling, we will approximate it to 7 months.

4. TASK SCHEDULE

The main tasks involving this project are:

1. Deliver the Requirement Analysis and Specification Document, containing the goals, the domain assumptions, and the functional and nonfunctional requirements of the software system.
2. Deliver the Design Document, containing the architecture and the design of the software system.
3. Deliver the Integration Testing Plan Document, containing the strategy used to perform integration testing on the system.
4. Deliver the Project Plan, which is this document.
5. Implement the software system and write unit tests.
6. Perform integration testing on the system.

The first four tasks for the project are already defined by the document about the project rules, together with the deadlines for the delivery of the RASD, the Design Document and the ITPD.

There are no fixed deadlines, instead, for the development of the software. From the COCOMO estimation performed in chapter 3, we expect the entire project to last 7 months, so it will be presumably finished by November 25 2016, assuming that the project has started in April 25 2016.

The schedule for our project is outlined in Table 4.1. The Gantt chart for the project is shown in Figure 4.1.

Activity	Start Date	Deadline
RASD	2016-04-25	2016-05-15
DD	2016-06-16	2016-05-29
ITPD	2016-06-13	2016-06-26
Project Plan	2016-06-27	2016-07-10
Implementation	2016-07-11	2016-11-09
Integration Testing	2016-11-10	2016-11-25

Table 4.1: Schedule for project tasks.

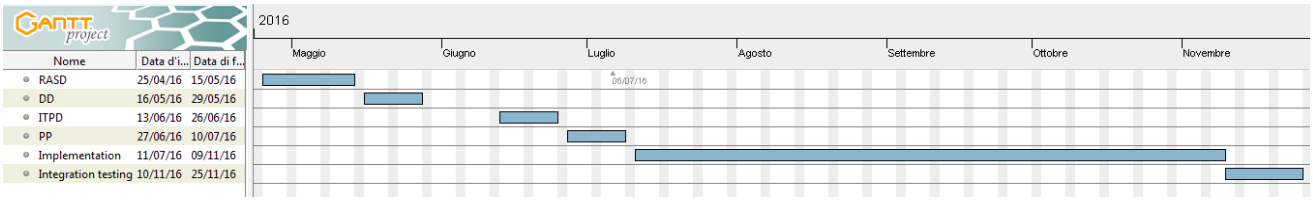

Figure 4.1: Gantt chart of the project.

5. RESOURCE ALLOCATION

5.1 RASD

Together [20 hours]	• Introduction • Functional Requirements • Specifications • Use Case description
Federico Dazzan [18 hours]	• Actors • Non Functional Requirements • Use Case diagram • Class diagram
Mirko Basilico[18 hours]	• Scenarios • Sequence diagrams • Alloy

5.2 DD

Federico Dazzan [24 hours]	• Introduction • Overview • Deployment view • Architectural styles and patterns • Other design decision • User Interface Design
Mirko Basilico[22 hours]	• Component view • Component interfaces • Runtime view • Algorithm Design
Together [14 hours]	• High level components and their interactions • Requirements Traceability

5.3 ITPD

Federico Dazzan [1 hour]	• Introduction • Tools and Test Equipment Required
Mirko Basilico [1 hour]	• Program Stubs and Test Data Required
Together [8 hours]	• Integration Strategy • Individual Steps and Test Description

5.4 PPD

Federico Dazzan [4 hours]	• Introduction • Task Schedule • Resource Allocation • Risk Management
Mirko Basilico [3 hours]	• Function Points analysis • COCOMO II analysis

5.5 IMPLEMENTATION AND INTEGRATION TESTING

Federico Dazzan [4 months – ESTIMATION]	• Front-end (Web and Mobile) development • Integration Testing
Mirko Basilico [4 months – ESTIMATION]	• Back-end development

6. RISK MANAGEMENT

PROJECT RISKS

The project risks are listed below:

Delays over the expected deadlines: the project could require more time than expected. If this happens, we may release a first, incomplete but working version (e.g. without the web interface or one of the features) and build the less essential features later.

Lack of communication among team members: the team often works remotely and this can lead to misunderstandings in fundamental decisions, to conflicts over the division of the work among team members and to conflicts in code versioning. Those difficulties can be overcome by explicitly defining (e.g. in this document) the responsibilities of each team member, and by writing clear and complete specification and design documents.

Lack of experience in programming with the specific frameworks: the team has no actual experience in programming using Java EE. This will certainly slow down the development.

Requirements change: the requirements may change during the development in unexpected way. This risk can't be prevented, but can be mitigated by writing reusable and extensible software.

Risk	**Probability**	**Effects**
Delays	High	Moderate
Lack of communication	Low	Moderate
Lack of experience	Certain	Moderate
Requirements change	Very low	Moderate

TECHNICAL RISKS

The technical risks are listed below.

Integration testing failure: after the implementation of some components, they may not pass the integration testing phase: this can require to rewrite large pieces of software. This risk can be mitigated by defining precisely the interfaces between components and subsystems, and by doing integration tests early using stubs and drivers.

Downtime: the system could go down for excessive load, software bugs, hardware failures or power outages. This risk can be mitigated by building redundant systems and placing them in geographically separate data centers and by performing testing at all levels.

Scalability issues: the system could not scale with a large number of users, requiring a major design rework. A possible plan is to use a cloud infrastructure from a third-party provider to host our system.

Spaghetti code: with the growth of the project, the code may become overloaded, badly structured and unreadable. This risk can be mitigated by writing a good Design Document before starting to code, and by performing code inspection periodically.

Deployment difficulties: The city TCC and DCC have to be integrated with our software, they need to provide the right APIs, this should be discussed in advance with the people in charge of these software systems

Data loss: data can be lost because of hardware failures, misconfigured software or deliberate attacks. This problem can be prevented by enforcing a reliable backup plan. Backups should be kept in a separate place from the system, and offline.

Data leaks: misconfiguration, software bugs and deliberate attacks can ex-pose the users' personal data. This risk must be prevented by adopting industry security standards, by encrypting communications and by doing regular penetration testing.

Risk	**Probability**	**Effects**
Integration testing failure	Low	High
Downtime	Moderate	High
Scalability issues	Low	Moderate
Spaghetti code	Low	Moderate
Deployment difficulties	Moderate	Catastrophic
Data loss	Low	Catastrophic
Data leaks	Moderate	Catastrophic

7. USED TOOLS

- **GanttProject:** to draw Task Gantt Diagram and Resource Allocation diagram

8. HOURS OF WORK

We have worked together most of the time and equally shared all the tasks and efforts. We have worked on this document for a total of 8 hours.

9. REFERENCES

Here is a short list of other references for this document:

- Slides of the Software Engineering 2 course (from the Beep Platform)