Politecnico di Milano

Software Engineering 2 project

PowerEnJoy

Integration Test Plan Document

Matteo Penco
mat. 875740

Riccardo Pressiani
mat. 874948

Version 1.0 approved

January 15, 2017

Revision History

Revision	Date	Author(s)	Description
0.1	03.01.17	RP	Document created
1.0	15.01.17	MP and RP	Document created

Hours of work

Matteo Penco	12h
Riccardo Pressiani	12h

Contents

1 Introduction **4**
- 1.1 Purpose . . . 4
- 1.2 Scope . . . 4
- 1.3 Definitions, acronyms and abbreviations . . . 5
  - 1.3.1 Definitions . . . 5
  - 1.3.2 Acronyms . . . 5
  - 1.3.3 Abbreviations . . . 5
- 1.4 References . . . 5
- 1.5 Overview . . . 5

2 Integration Strategy **6**
- 2.1 Entry Criteria . . . 6
- 2.2 Elements to be Integrated . . . 6
- 2.3 Integration Testing Strategy . . . 7
- 2.4 Sequence of Component/Function Integration . . . 8
  - 2.4.1 Test Plan 1 . . . 8
  - 2.4.2 Test Plan 2 . . . 8
  - 2.4.3 Test Plan 3 . . . 8
  - 2.4.4 Test Plan 4 . . . 9
  - 2.4.5 Test Plan 5 . . . 9
  - 2.4.6 Test Plan 6 . . . 10
  - 2.4.7 Test Plan 7 . . . 10
  - 2.4.8 General Overview . . . 11

3 Individual Steps and Test Description **12**

4 Tools and Test Equipment Required **17**
- 4.1 Tools . . . 17
- 4.2 Test Equipment . . . 17

5 Program Stubs and Test Data Required **19**
- 5.1 Program Stubs and Drivers . . . 19
- 5.2 Test Data . . . 19
  - 5.2.1 Data Set Test Case I1 . . . 19

5.2.2 Data Set Test Case I2 20
5.2.3 Data Set Test Case I3 20
5.2.4 Data Set Test Case I4 20
5.2.5 Data Set Test Case I5 20
5.2.6 Data Set Test Case I6 20
5.2.7 Data Set Test Case I7 21
5.2.8 Data Set Test Case I8 21
5.2.9 Data Set Test Case I9 21
5.2.10 Data Set Test Case I10 22
5.2.11 Data Set Test Case I11 22
5.2.12 Data Set Test Case I12 23
5.2.13 Data Set Test Case I13 23
5.2.14 Data Set Test Case I14 23

List of Figures **24**

List of Tables **24**

Bibliography **26**

Chapter 1

Introduction

1.1 Purpose

This document provides a description of the PowerEnJoy software integration test plan. The integration testing is a key phase of software lifecycle: it allows the software development project manager to organize and plan parallel activities for the team and to test step by step the interactions among the developed software blocks. The purpose of this document is to outline the main aspects of the strategy, procedures and expected results of the integration testing phase. This document is intended for the quality improvement team, that includes both the quality assurance and development team.

1.2 Scope

PowerEnJoy is inteded to be a management system for a car-sharing system that exclusively employs electrical powered cars. The system allows users to find all the available cars near a given location which can be their current position or a specific address typed in. The user can book one of the cars available for a limited amount of time. Once the booked car is reached by the user, it can be unlocked from one of the smart devices of the user. After entering a PIN, decided by the user during the registration phase, on the onboard computer, the engine can be started and the rental begins. The cost of the service is calculated on the total duration of the rental multiplied by a fixed rate per minute. The user is continuously informed about the cost of the ongoing rental by the onboard computer. The user can end the rental by parking the car and stopping the engine.

The PowerEnJoy platform is intended to be available on the major mobile devices, such as smartphones and tablets running iOS [1] or Android [2].

1.3 Definitions, acronyms and abbreviations

1.3.1 Definitions

1.3.2 Acronyms

DD Design Document

ORM Object-Relational Mapping

RASD Requirement Analysis and Specification Document

PIN Personal Identification Number

1.3.3 Abbreviations

App Application

1.4 References

- Assigments document for the first semester project of the Software Engineering 2 course held at Politecnico di Milano by Mottola Luca and Di Nitto Elisabetta [3].
- PowerEnjoy - Requirement Analysis and Specification Document [4].
- PowerEnjoy - Design Document [5].

1.5 Overview

A general introduction about the PowerEnJoy project has been given in this chapter. In the following chapters the reader will be provided with a more detailed description of the integration testing procedures.

In Chapter 2, the entry criteria to begin the integration testing phase will be provided. Moreover, the strategy and the integration test order will be described.

In Chapter 3, a detailed description of each integration test will be provided. These descriptions will include the expected input values and the corresponding state after each method invocation.

In Chapter 4, tools and equipment used to perform the tests will be listed along with a brief description.

In Chapter 5, the drivers needed and the set of data used to correctly perform the tests will be described.

Chapter 2

Integration Strategy

2.1 Entry Criteria

Before starting the integration testing phase, there are some requirements that need to be met. It is fundamental to be completely aware of the software to be and some guarantees about the work done by developers on the single components must be assured.

First of all, both the Requirements Analysis and Specification Document [4] and the Design Document [5] must be fully redacted and approved. The RASD is essential for the developer team to let them develop the requested functionalities in a correct way. Moreover, the integration testing cannot begin unless all the components to be integrated are completely unit tested. The unit testing phase is feasible only if the software requirements are fully redacted and detailed. The DD (with respect to Chapter 2) presents the general software architecture and the interactions among components. This information is fundamental to design a correct and complete integration testing sequence, which will be described later in this chapter.

Finally, the database schema needs to be designed before the integration testing and a instance of the database must be running locally or remotely, based on where the integration tests are executed. This allows the development team and especially those in charge of the integration testing to generate and maintain an instance of the system state during the tests execution.

2.2 Elements to be Integrated

In this section a list of all the components that need to be integrated is provided. We remind the reader that a complete view of the system components is provided in Section 2.3 of the Design Document.

First of all, the following components are the ones that together compose the core of the PowerEnJoy system:

- UserController
- CarController
- MapController
- BookingController
- RentalController

Some of the functionalities implemented in the PowerEnJoy system are provided by external services. Helper components, such as PaymentHelper, NotificationHelper and RecoveryHelper, are in charge of the communications between the system and these third party services. Since the goal of these component is only to exchange formatted messages with the external services, the effort for their integration tests will be quite limited.

All the controllers that provide methods that can be called by the client applications need to be integrated with the Router component. The Router is the component in charge of receiving requests from the clients and to execute the right routines on the related controllers.

For what concerns the communication between the system and the database, an external library is used as an ORM.

2.3 Integration Testing Strategy

It is important to follow a pre-determined strategy for the integration testing to save costs and time. If the integration testing is performed for the first time on the entire system, the risk of allocating more resources than the ones expected can be very high.

For this reason, the strategy adopted for the PowerEnJoy system is mainly bottom-up with some components integrated as threads. Moreover, a critical-module-first logic is considered in the whole process to assign priorities to the different test plans.

In a bottom-up integration strategy, components need to be fully developed and unit tested to be integrated with other components. This allows to test incrementally small portions of the software. Thus, it will be more likely to find errors and faults in the early stages of the process. Moreover, components without dependencies among them can be tested in parallel and this is essential to maximize time and resources.

Components related to the communications between the system and external services can be integrated at any stage of the development process. For this reason, the strategy that will be adopted is by threads: the integration test of these specific components will be executed when the functionalities are implemented. Other integration tests related to the parent component can be performed without dependencies to components thread-integrated.

2.4 Sequence of Component/Function Integration

In this section we are going to describe the order of integration of PowerEnJoy's components. As already mentioned, the adopted integration strategy is mainly bottom-up, so, as notation, an arrow going from component C1 to component C2 means that C1 must have been already developed and unit tested in order to be integrated with C2.

2.4.1 Test Plan 1

Referring to the critical-module-first logic adopted in the integration strategy, CarController is the most important component in the system, so we start to integrate it with the other related components.

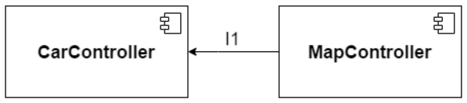

Figure 2.1: The picture above shows the Test Plan 1 diagram. In this Test Plan, MapController is integrated with CarController.

2.4.2 Test Plan 2

For the same reasons explained above in Test Plan 1, we can integrate RecoveryHelper with CarController using the threads strategy explained above.

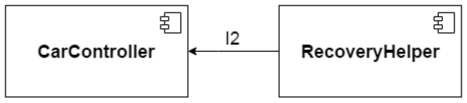

Figure 2.2: The picture above shows the Test Plan 2 diagram. In this Test Plan, RecoveryHelper is integrated with CarController.

2.4.3 Test Plan 3

The core of the system is surely the relation between CarController, Booking-Controller and RentalController, so the next step is to integrate CarController with the other two components.

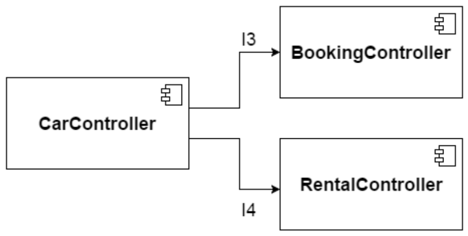

Figure 2.3: The picture above shows the Test Plan 3 diagram. In this Test Plan, CarController is integrated with BookingController and RentalController.

2.4.4 Test Plan 4

We can now integrate RentalController with BookingController to have core functionalities of the system fully tested and integrated.

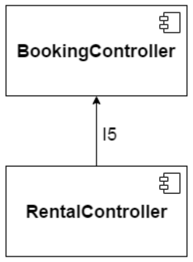

Figure 2.4: The picture above shows the Test Plan 4 diagram. In this Test Plan, RentalController is integrated with BookingController.

2.4.5 Test Plan 5

By threads we can integrate PaymentHelper with BoookingController and Rental-Controller, following the bottom-up strategy in order to test the full system.

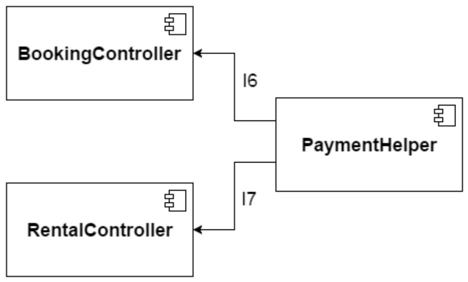

Figure 2.5: The picture above shows the Test Plan 5 diagram. In this Test Plan, PaymentHelper is integrated with BookingController and RentalController.

2.4.6 Test Plan 6

For the same reasons explained above in Test Plan 5, we can also integrate by threads NotificationHelper with CarController and BookingController.

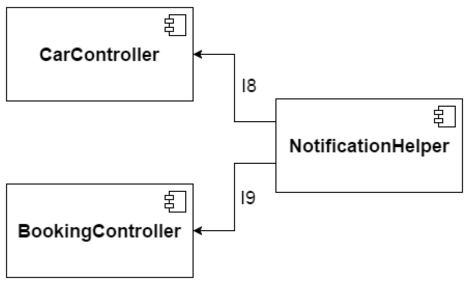

Figure 2.6: The picture above shows the Test Plan 6 diagram. In this Test Plan, NotificationHelper is integrated with CarController and BookingController.

2.4.7 Test Plan 7

Finally we can test the integration between all core components and the Router component, which is needed to start every process/request in the system.

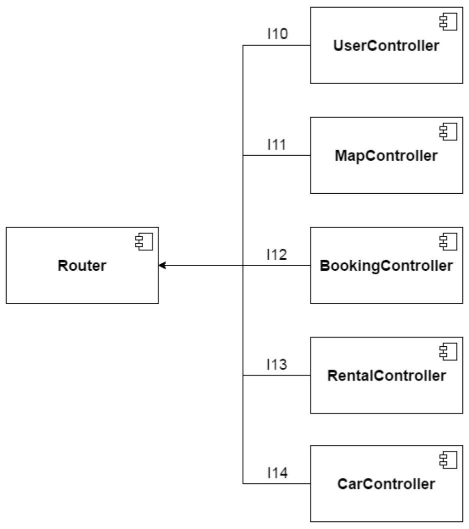

Figure 2.7: The picture above shows the Test Plan 7 diagram. In this Test Plan, all core components are integrated with the Router component.

2.4.8 General Overview

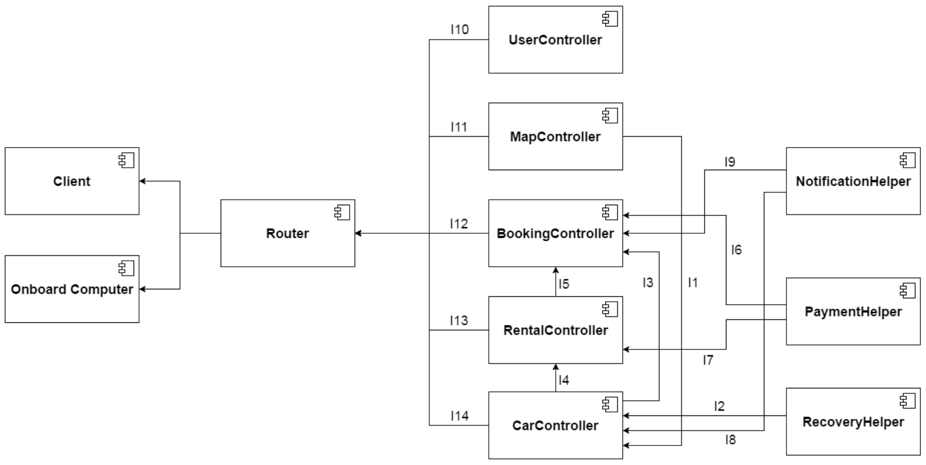

Figure 2.8: The picture above shows the general overview of how all components are integrated in PowerEnJoy's system.

Chapter 3

Individual Steps and Test Description

In this chapter a detailed description of each integration test will be provided. For each integration between components C1 and C2 we are going to test all methods that C1 invokes on C2, using this notation: C1 $\rightarrow$ C2. For each method we are going to provide the expected input values and the effect after the invocation.

Test Case Identifier	I1
Test Items	CarController $\rightarrow$ MapController
checkCarInSafeArea(CarPosition)	
Input Specification	A valid car position.
Output/Effect	Returns true if the car is parked in a safe area, false otherwise.

Test Case Identifier	I2
Test Items	CarController $\rightarrow$ RecoveryHelper
notifyRecoveryService(CarStatus)	
Input Specification	A valid car status.
Output/Effect	A notification containing information about the car is sent to the recovery service.

Test Case Identifier	I3
Test Items	BookingController → CarController
	updateCarStatus(Car)
Input Specification	A valid car.
Output/Effect	Car status is correctly updated depending on the current situation.

Test Case Identifier	I4
Test Items	Rental → CarController
	updateCarStatus(Car)
Input Specification	A valid car.
Output/Effect	Car status is correctly updated depending on the current situation.

Test Case Identifier	I5
Test Items	BookingController → RentalController
	startRental()
Input Specification	No input is required for this method.
Output/Effect	A new rental instance is created.

Test Case Identifier	I6
Test Items	BookingController → PaymentHelper
	notifyPaymentService(bill)
Input Specification	A valid bill.
Output/Effect	The bill the user has to pay, depending on the situation, is sent to the payment service.

Test Case Identifier	I7
Test Items	RentalController → PaymentHelper
	notifyPaymentService(bill)
Input Specification	A valid bill.
Output/Effect	The bill the user has to pay, depending on the situation, is sent to the payment service.

Test Case Identifier	I8
Test Items	CarController → NotificationHelper
	notifyUnlockCar(car)
Input Specification	A valid car.
Output/Effect	The user is notified of car being unlocked.

Test Case Identifier	I9
Test Items	BookingController → NotificationHelper
	notifyElapsedBooking(booking)
Input Specification	A valid booking.
Output/Effect	The user is notified of his/her booking being elapsed.

Test Case Identifier	I10
Test Items	Router → UserController
	sendRegistration(form)
Input Specification	A valid registration form.
Output/Effect	The new user is registered and inserted in the database.
	sendLogin(form)
Input Specification	A valid login form.
Output/Effect	The user is successfully logged.
	sendPin(form)
Input Specification	A valid PIN form.
Output/Effect	Car's engine can now be ignited.

Test Case Identifier	I11
Test Items	Router → MapController
	findCars(position)
Input Specification	A valid position.
Output/Effect	All available cars near the provided position are shown.
	findSafeAreas(position)
Input Specification	A valid position.
Output/Effect	All safe areas near the provided position are shown.

Test Case Identifier	I12
Test Items	Router → BookingController
	confirmBooking(car)
Input Specification	A valid car.
Output/Effect	The status of the car selected by the user is set to RESERVED.
	bookingToRental()
Input Specification	No input is required for this method.
Output/Effect	A new rental request is sent to the BookingController in order to start the rental.

Test Case Identifier	I13
Test Items	Router → RentalController
	getInfo()
Input Specification	No input is required for this method.
Output/Effect	Information about the ongoing rental is shown to the user.
	endRental(carStatus)
Input Specification	A valid car status.
Output/Effect	A request is sent to the RentalController in order to end the ongoing rental.

Test Case Identifier	I14
Test Items	Router → CarController
	getCarDetails(car)
Input Specification	A valid car.
Output/Effect	Information about the selected car is shown to the user.
	checkCarPosition()
Input Specification	No input is required for this method.
Output/Effect	The current position of the car is calculated.
	unlockCar(car)
Input Specification	A valid car.
Output/Effect	The car is unlocked.

Chapter 4

Tools and Test Equipment Required

4.1 Tools

In order to perform the integration testing in an efficient way both automated tools and manual testing will be used.

The automated part of the integration testing will include all the components of the backend and will be done using the jUnit framework. jUnit is a valid tool to verify that the software procedures execute correctly if the correct input data are provided. Moreover, if something fails it assures that all the exceptions are caught and managed correctly.

In addition, an important phase of the integration testing will be devoted to the manual testing of the client applications, that include the web app, the mobile app and the onboard computer app. Communications with the backend must be tested and the correct execution of all the procedures must be guaranteed.

4.2 Test Equipment

Both client applications and the backend infrastructure must be tested in environment similar to the deployment ones. In this section the testing environments that have been chosen to perform the integration testing are provided.

Several mobile devices will be tested with the mobile applications and the mobile version of the web app:

- iOS
  - Each iPad starting from the 3rd generation one
  - Each iPhone starting from the 4S model

- Android
  - At least one smartphone from 4” to 6” at steps of 1/2”
  - At least one tablet from 7” to 12” at steps of 1”

The desktop web application will be tested with the most common browsers (Safari, Chrome, Firefox and Edge) and displays that vary from 11” to 27”. No specific hardware requirements are needed.

The onboard computer hardware will be unique and deployed on every car in the system. For this reason a single device with a fixed display will need to be tested.

The backend will be tested using a cloud infrastructure with the same topology of the one that will be used in production. The hardware requirement of the development and testing environment will be limited with respect to the production environment because a smaller amount of resources is needed during the testing phase.

Chapter 5

Program Stubs and Test Data Required

5.1 Program Stubs and Drivers

Since the strategy adopted for the integration testing phase is bottom-up, no stubs are required to be implemented.

However, two drivers are expected to be developed in order to perform the integration testing correctly. The first one is needed to create different car states in order to test the integration among the CarController, the MapController and the RecoveryHelper. The second one is needed to simulate the users behaviour once the BookingController and the RentalController are being tested.

5.2 Test Data

In this section we provide the set of possible data to be considered in order to correctly and totally test the integration between components.

5.2.1 Data Set Test Case I1

checkCarInSafeArea(CarPosition)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A CarPosition whose coordinates are invalid.	An InvalidPositionException is raised.

5.2.2 Data Set Test Case I2

notifyRecoveryService(CarStatus)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A CarStatus with invalid fields.	An InvalidArgumentValueException is raised.

5.2.3 Data Set Test Case I3

updateCarStatus(Car)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.

5.2.4 Data Set Test Case I4

updateCarStatus(Car)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.

5.2.5 Data Set Test Case I5

startRental()	
Input	*Effect*
No input is required for this method.	

5.2.6 Data Set Test Case I6

notifyPaymentService(bill)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A bill with invalid fields.	An InvalidArgumentValueException is raised.

5.2.7 Data Set Test Case I7

notifyPaymentService(bill)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A bill with invalid fields.	An InvalidArgumentValueException is raised.

5.2.8 Data Set Test Case I8

notifyUnlockCar(car)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.

5.2.9 Data Set Test Case I9

notifyElapsedBooking(booking)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A booking with invalid fields.	An InvalidArgumentValueException is raised.

5.2.10 Data Set Test Case I10

sendRegistration(form)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A form with invalid fields.	An InvalidArgumentValueException is raised.
sendLogin(form)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A form with invalid fields.	An InvalidArgumentValueException is raised.
sendPin(form)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A form with invalid fields.	An InvalidArgumentValueException is raised.

5.2.11 Data Set Test Case I11

findCars(position)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A position whose coordinates are invalid.	An InvalidPositionException is raised.
findSafeAreas(position)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A position whose coordinates are invalid.	An InvalidPositionException is raised.

5.2.12 Data Set Test Case I12

confirmBooking(car)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.

bookingToRental()	
Input	*Effect*
No input is required for this method.	

5.2.13 Data Set Test Case I13

getInfo()	
Input	*Effect*
No input is required for this method.	

endRental(carStatus)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.
A CarStatus with invalid fields.	An InvalidArgumentValueException is raised.

5.2.14 Data Set Test Case I14

getCarDetails(car)	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.

checkCarPosition()	
Input	*Effect*
No input is required for this method.	

unlockCar(car))	
Input	*Effect*
A null parameter.	A NullArgumentException is raised.

List of Figures

2.1 Test Plan 1 diagram 8
2.2 Test Plan 2 diagram 8
2.3 Test Plan 3 diagram 9
2.4 Test Plan 4 diagram 9
2.5 Test Plan 5 diagram 9
2.6 Test Plan 6 diagram 10
2.7 Test Plan 7 diagram 10
2.8 General Overview diagram 11

List of Tables

3.1 Test Case 1 12
3.2 Test Case 2 12
3.3 Test Case 3 13
3.4 Test Case 4 13
3.5 Test Case 5 13
3.6 Test Case 6 13
3.7 Test Case 7 14
3.8 Test Case 8 14
3.9 Test Case 9 14
3.10 Test Case 10 14
3.11 Test Case 11 15
3.12 Test Case 12 15
3.13 Test Case 13 15
3.14 Test Case 14 16

5.1 Data Set Test Case I1 19
5.2 Data Set Test Case I2 20
5.3 Data Set Test Case I3 20
5.4 Data Set Test Case I4 20
5.5 Data Set Test Case I5 20
5.6 Data Set Test Case I6 20
5.7 Data Set Test Case I7 21
5.8 Data Set Test Case I8 21
5.9 Data Set Test Case I9 21
5.10 Data Set Test Case I10 22
5.11 Data Set Test Case I11 22
5.12 Data Set Test Case I12 23
5.13 Data Set Test Case I13 23
5.14 Data Set Test Case I14 23

Bibliography

[1] Apple Inc. iOS. Available at `http://www.apple.com/ios/` (November 26, 2016).

[2] Google Inc. Android. Available at `https://www.android.com/` (November 26, 2016).

[3] L. Mottola and E. Di Nitto. *Assignments AA 2016-2017*. Politecnico di Milano.

[4] M. Penco and R. Pressiani. PowerEnjoy - Requirement Analysis and Specification Document. Available at `https://github.com/rpressiani/se2-project-2016/blob/master/releases/RASD.pdf` (January 03, 2016).

[5] M. Penco and R. Pressiani. PowerEnjoy - Design Document. Available at `https://github.com/rpressiani/se2-project-2016/blob/master/releases/DD.pdf` (January 03, 2017).