Inf2C Software Engineering 2015-16

Coursework 1

Capturing requirements for a city bike-hire scheme

1 Introduction

The aim of this coursework is to produce a requirements document for the software part of a new selve-serve bike-hire scheme for Edinburgh. Later courseworks will involve the design and implementation of this software.

2 System Description

Background

Self-serve bike-hire schemes already operate in many cities around the world. Such schemes provide cities with an alternative healthy environmentally-friendly mode of transport. They are useful for when city dwellers don't want to own, maintain and store their own bike, for when travel is combined with other modes of transport, and for visitors to a city. The proposed scheme is a simplified version of the scheme used in London [1].

Docking stations

Spread around appropriate locations in the city are docking stations, each consisting of a terminal and a number of docking points for the bikes. The terminal includes a touch-screen display and a bank-card reader.

Registering

Before hiring a bike, users must first register at a docking station terminal. They do so by entering a few personal details at a docking station terminal and inserting a credit or debit

[1] `https://en.wikipedia.org/wiki/Santander_Cycles`

card into the bank-card reader. Assuming the card is valid and a correct PIN is entered, the terminal then issues the user with a contactless electronic key.

Hiring and returning bikes

To hire a bike, a user inserts their key into a key-reader slot at at some occupied docking point and immediately removes the key. The docking point unlocks the bike and a green OK light on the docking point flashes.

To return a bike at some docking station, the user inserts the bike into some free docking point at the station. The point detects the bike has been inserted, locks the bike and flashes the OK light.

A user can hire at most one bike at a time with a given key.

Charges

Hire charges are based on the length of hire: the charge is £1 for hires of up to 30 min., and then £2 extra for time in each further 30 min. period. The higher charge for subsequent periods is to encourage short frequent use of the bikes. Each docking point can read a unique ID of any bike inserted into it, so there is no need for users to otherwise identify themselves when returning a bike.

Charges are made to users' bank cards at midnight for all trips completed in the previous 24 hours. To check on charges accumulated through the day, users can request a docking station terminal to display details of their trips, the time and location of each trip start and end and the charge for each of the trips. for example.

When hire goes wrong

Bike hire doesn't always flow completely smoothly.

For example, on attempting to return a bike to a docking station, it may turn out that all docking points are occupied. In this case a user can identify themselves at the terminal using their key, and the terminal display shows a map of streets nearby which has marked on it locations of other docking stations and the number of free points at each. The system gives the user a 15 min. extension on the current 30 min. period for them to have time to reach one of these alternate stations.

Bikes sometimes develop faults. If so, a user should return the bike to a station nearby and press the *fault* button at the docking point. A red light on the docking point then comes on to warn that the docked bike is out of order. For the fault button to work, it must be pressed within 10 sec. of the bike being docked.

Supervising operations

The scheme is supervised from an Operations Hub which runs the Hub System. This system has wireless data-links with all the docking stations. A large wall display at the hub shows the status of the stations. Information shown includes the locations where docking point occupancy is under 15% or over 85%, and where there are bikes with reported faults.

Personnel at the hub coordinate the movement of bikes to docking stations where bikes are running out, and to and from a bike service centre. Staff taking care of these movements are issued with keys for unlocking bikes. The software is not responsible for planning and advising on how bikes should best be moved and when bikes should be picked up for service.

The overall configuration of the system is managed at the hub. For example, when a new station is added, an operator of the hub system registers the station with the hub system. Docking station terminals track their docking points as they are electronically connected to the terminal.

On demand the hub system can also generate reports, showing statistics on the recent use of the system, and indicating any bikes that have been hired but not returned in over 24 hours.

3 Your task

Your job for this coursework is to create a requirements document for the software.

Start your document with a few sentence description of the whole system and then refer the reader to these instructions for further general information. There is no need to reproduce the description here. Then include in your document the information asked for in the following tasks.

3.1 Choose a system name

Devise a snappy name for your system!

3.2 Identify inputs and outputs

Systematically identify the inputs and outputs of the whole software system, i.e. buttons, displays, sensors, lock actuators, etc. that it connects to. Treat the hub system, all the docking terminals, and all the docking points as a single system. In practice, the software system is a distributed system, with separate computers for the hub system and each docking station. For simplicity, when designing and implementing the system, you will ignore the distributed nature of the system. Here in the requirements documents just acknowledge the connections between the main system components. Do make clear in your listing of each input and output device whether it connects to the hub system, a docking station terminal or a docking point.

3.3 Catalog requirements

Describe the requirements of the system broken down as an enumerated list of individual requirements, sometimes known as *features*, of the software system. Each requirement should be reasonably short and describe a single aspect of the system. As appropriate, you can use two or more layers of enumeration: for example, you may have a requirement numbered 2.a.ii.

Include functional requirements that might not be captured by use cases and a few main non-functional requirements.

These requirements essentially will form a contract for what is expected of the software system. You should imagine that at some stage the transport section of the Edinburgh council will officially agree with you that that these requirements are what they want. You might also check these requirements satisfy other stakeholders too. These requirements will later form the basis of the system tests: at least one test will have to be devised to show that each requirement has been met.

3.4 Describe use-cases

Identify a representative set of perhaps 8-10 use cases for the system. Keep the use cases high level: they are about the main interactions between actors external to the system and the system itself, they should not be concerned with all the details of user interface interactions: you are not doing design at this stage.

Do not try to capture all possible processing scenarios with your use-cases. Focus on the main processing to clarify what required functions are needed. This is typically enough for the customer to understand the system and for you to understand the customer's needs.

These use cases will help clarify requirements and will be of use later when checking whether your design is reasonable and when creating tests.

Produce a description of each use case using a template similar to that described on the *Tutorial 1* question sheet used for the Week 4 tutorials. Feel free to add extra fields if you feel it would help, and also omit fields when they are unnecessary. For example, if the *Main Success Scenario* is one step that is obvious from the *Guarantee*, there is no need to explicitly describe it further. Do add a field identifying which requirements a use case exercises, and do number all your use cases for ease of reference.

Draw a UML use-case diagram summarising your use cases and the actors you have identified for each.

4 Asking Questions

Please ask questions on the class discussion forum if you are unclear about any aspect of the system description or about what exactly you need to do. For this coursework tag your questions using the *cw1* folder. As questions and answers build up on the forum, remember to check over the existing questions first: maybe your question has already been answered!

5 Submission

Please submit two files

1. A PDF (not a Word or Open Office document) of your requirements document. The document should include **a title page with names and UUNs of the team members**.

2. a text file named **team.txt** with only the UUNs of the team members (one UUN on each line) as shown,

```
s1234567
s7891234
```

How to Submit

Ensure you are logged onto a DICE computer and are at a shell prompt in a terminal window. Place your PDF requirements document and your team.txt file in a directory called `Application`. Then ensure the current directory is set to the directory containing this `Application` directory and submit your work using the command:

```
submit inf2c-se 1 Application
```

This coursework is due

Noon on **Friday 23rd October**

The coursework is worth 20% of the total coursework.

Paul Jackson, 15th October 2015.