Inf2C Software Engineering 2016-17

Coursework 1

Capturing requirements for a restaurant order-management system

1 Introduction

The aim of this coursework is to produce a requirements document for the software part of a system for managing food and drink orders in a restaurant. Later courseworks will involve the design and implementation of this software.

2 System Description

Background

Traditionally, in restaurants providing table service, a waiter takes the food order of a set of customers at a table on a piece of paper called a *ticket*, usually making one or more carbon copies. One ticket copy goes to the kitchen and is put up on a rack for the chefs and other kitchen staff to read. Another ticket copy is handed to the cashier. Once the food and drinks are ready, they are passed out with the ticket for the waiter to deliver to the table which placed the order. Drinks might be handled in a similar way, with a separate ticket going to the bar. When a table is ready to pay, the waiter gathers together associated ticket copies that have been lodged with the cashier and presents the table with the total bill. The customers at the table then pay the bill by some means, e.g. cash or credit card.

In recent years, it has become common for automated support to be provided for the handling of tickets by the cashier and in the kitchen. In this coursework the aim is to explore the software requirements for a simplified order management system which not only does this but also takes care of the order taking and bill paying. Below is a description of how the system is to handle food and drink orders. The description assumes that both food and drink orders are handled in the kitchen. In practice, drinks orders might be handled separately at a bar.

Order taking

Each table is equipped with a touch-screen display which presents food and drinks menus. Customers at the table are able to interact with the menu, building an order of food and drinks. The building process involves the display showing the current state of the order, allowing new items to be added and existing items to be removed. When the order is ready, one of the customers hits the 'order submit' button.

Order management in the kitchen

A large touch-screen display in the kitchen presents the current food orders to the chefs, showing for each order the items selected and quantities, along with the time since the order was submitted. Orders are listed in order of submission time.

When an item is ready, the chef who had prepared it places it in an area called the *pass*, ready for a waiter to pick up. The chef also indicates on the touch-screen display the item is ready. When the first item of an order is indicated ready, the system prints an order ticket which is placed in the pass along with the item. Subsequent items placed in the pass are then grouped with this first item and the ticket.

Food and drinks delivery

By the kitchen pass is an *order-up* light which is lit when an order is complete and up on the pass, ready for collection. Waiters keep an eye on the order-up light, and, when an order is ready for pickup, they collect it and deliver it to the table that placed the order. A push button by the pass allows waiters to turn off the order-up light if no further orders are ready when they pick up.

Bill paying

Customers request their bill using the touch-screen display at their table. The display shows the full list of items ordered and the total amount due. For simplicity, let's assume one of the customers at a table pays the full bill and pays by just tapping a contactless smart card against the table display. Assume a connection is made over the internet to some bank server in order to get authorisation for payment. Each display unit also has a built-in mini printer that can print out a receipt, if the customer asks for one.

Behind the scenes

In the restaurant office is a computer with keyboard, mouse and display at which the head chef enters and updates the food and drinks menus. The restaurant manager can also use this computer to view a summary of order activity over specified periods, e.g. the past day or week. From day to day, the restaurant table layout may change, and tables are added or taken away. The system must cope easily with this.

Extensions

The above has just sketched out the basics. In real life many refinements would be possible and desirable. A few are sketched below. There is no need to cover any of these in the requirements document you produce.

- Orders are usually split into parts, e.g. drinks, starters, main courses, desserts that are scheduled for arrival at different times.
- Different order parts will be fulfilled in different places. The drinks at the bar, hot food, cold food, desserts in different areas of the kitchen. Each could have a separate display for orders needing fulfilling.
- Customers might order extra items after submitting their initial order.
- The table displays could indicate to customers when their orders are expected to be ready.
- The kitchen display could use knowledge about how long items take to prepare and could recommend scheduling of items in an order such that all items become ready together.

3 Your job

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 Identify stakeholders

Identify the stakeholders of the system and the value or benefit each seeks from it.

3.2 Describe the physical architecture

List the various input, output and/or input-output devices of the system by their locations in the restaurant. Indicate for each its kind (*input*, *output* or *input-output*).

For convenience, refer to the keyboard, mouse and display for the office computer simply as the *office computer* input-output device and also treat touchscreen display just as an input-output device (i.e. there is no need to separate it into a touch input device and screen display output device).

In practice, the software system would be realised as a distributed system, with separate computers for each of the tables, perhaps also some distinct central computer distinct from the office computer, and all the computers connected by a wired or wireless network.

In subsequent design and implementation courseworks, this distributed nature will be largely ignored for simplicity. Here too for this requirements document there is no need to say anything specific about the internal architecture of the system.

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.

Be sure to 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 restaurant manager will officially agree with you that these requirements are what are wanted. 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.

Add a note of any details that are obviously missing from the provided system description, and if there is anything that is ambiguous and needs further clarification.

It is not necessary to insert into the functional requirements significantly more detail than there is in the system description above. Rather, what is being looked for is good organisation of the requirements, with related requirements grouped together.

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.

For three of the use-cases produce a description 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. For the rest of the use-cases, a simpler format with just the primary actor and a free-text description is fine.

Do number all your use cases for ease of reference, and do add a field for each identifying which requirements it exercises,

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 Good Scholarly Practice

Please remember the University requirement as regards all assessed work for credit. Details and advice about this can be found at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

and links from there. Note that, in particular, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository, then you must set permissions so that only members of your coursework group can access your work.

6 Marking scheme

Marks will be allocated as follows.

- 10% Stakeholders and their interests
- 10% Physical architecture description
- 35% Requirements
- 10% Use case diagram
- 35% Use cases

7 Submission

Please submit two files

1. A PDF (not a Word or Open Office document) of your requirements document. The document should be named **report.pdf** and 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

Only one member of each coursework group should make a submission. If both members accidentally submit, please alert the course organiser so confusion during marking is avoided.

Ensure you are logged onto a DICE computer and are at a shell prompt in a terminal window. Place your *report.pdf* requirements document and your *team.txt* file in the same directory, ensure this directory is set as your current directory (i.e. `cd` to it), and submit your work using the command:

```
submit inf2c-se 1 report.pdf team.txt
```

This coursework is due at

16:00 on **Thursday 20th October**

The coursework is worth 20% of the total coursework mark and 8% of the overall course mark.

Paul Jackson, 10th October 2016.