

# Inf2C-SE 2014/2015

Issued on: Thursday 18<sup>th</sup> September, 2014

If you attempt some of this *before* your tutorial then your time at the tutorial may be more beneficial, spent discussing your solutions with your fellow students and tutors. Alternatively these exercises can be done within your tutorial. In both cases working in pairs or groups is encouraged.

## 1 Introduction

In this tutorial you will look at some example scenarios and design some use-cases, use-case diagrams and requirements for the proposed systems. There are a number of proposed systems, you should not feel that you need to tackle all of these. It is more important that you complete one or two well, rather than briefly attempt each one. The first two are rather gentle and the others are more complex. A good exercise would be to complete one of the first two and then make an attempt at the more complex systems.

If there is time at the end of the tutorial, there is a more advanced discussion section. Attempt this with your tutor if you are confident of having understood the earlier parts.

The following two sections present templates for use-cases and individual requirements specifications. You should also attempt to present the use-case diagram for each system that you attempt.

## 2 Use-Case Template

**Use Case Name** The name of the use case

**Iteration** Use cases typically evolve in a series of iterations. For example, first we do a draft version, then we fill out the exact interactions, and, finally, we wrap up all the details.

**Summary** A one or two sentence description of the use case. Do not just reiterate the steps in the course of events section.

**Basic Course of Events** This is the meat of the use case. Here you record the steps the user would go through to achieve the goal. Focus on the most common and simplest course of events, if there are alternative ways, they can be recorded below or in a separate use case.

**Alternative Paths** If there are some alternative ways of achieving the use case, this can be recorded here (or in a separate use case).

**Exception Paths** Here we record what to do if there are errors, for example, incorrect input. This can also be recorded in separate use cases.

**Extension Points** If this use case is an extension of some other use case (or vice versa), this relationship can be recorded here.

**Trigger** The event that leads to this use case being performed.

**Assumptions** Any assumptions you make about the environment or user for this use case to work correctly.

**Precondition** The contract between this use case and the outside world. All things that are outside the scope of the system, but are required to be there for the use case to work.

**Postcondition** What will hold after this use case is completed.

**Author** Who wrote it down?

**Date** When that person wrote it down.

### 3 Individual Requirements Template

**Requirement Number** A unique number

**Requirements Type**

**Use Case** The use case it is associated with

**Introduction** A short description about what this requirement is for

**Rationale** Why is the requirement here?

**Author** Who wrote down this requirement specification.

**Source** Who came up with the requirement.

**Inputs** The inputs needed to perform this user function.

**Requirement Description** The actual requirement expressed in the users language.

**Outputs** The outputs generated as a result of this user function.

**Persistent Changes** Database updates or other persistent changes.

**User Satisfaction** Often presented on a scale of 1-5

**User Dissatisfaction** Similarly presented on a scale of 1-5

**Related Requirements** Cross reference to other requirements on which this one relies.

**Conflicts** There may be conflicts in the document. Which requirement can we not implement if we implement this one.

**Support Materials** Any documents, graphs, etc., that are related to this requirement.

**Test Cases** Test cases that can be used to verify that this requirement is implemented correctly. Must include both inputs and expected outputs.

### 4 Example Systems

In this section several example systems are presented. Do as many of these as you wish, but you should focus on finishing one entirely rather than doing many to a very narrow depth. The first two are pretty simple systems, so if you are unsure try one of these first and then proceed to the other more involved systems.

## 4.1 Thermostat - Basic

A thermostat is a simple system to control the temperature in a heated home. The user sets the desired temperature, if any (monitored) room in the house is below that temperature the heating is turned on. When all monitored rooms are above the desired temperature the heating is turned off.

Design some use-case and requirements specifications to handle this system. Take note of what happens when the user first starts the system, and also when the user changes the desired temperature.

## 4.2 Elevator - Basic

We have all used elevators before. Consider a single elevator system, and simply consider the scenario from the perspective of a user wishing to use the elevator. In other words, do not worry about the scenario of multiple simultaneous users. So we are simply looking at the functionality of a call button. The user presses a button indicating in which direction they intend to travel in the elevator. At some point the elevator should arrive at the floor to collect the waiting user.

A small part of this is that the call button should indicate to the user whether or not the elevator is coming. Generally this involves some kind of light embedded into the button. This is useful feedback to the user such that they know their request has been registered and they have not been dismissed/forgotten about. Hint: to have any effect, this light needs to be extinguished when the current request is met.

As an extension you could add a ‘cancel’ button. If a user has waited too long they may instead decide to use the stairs and may cancel their request.

## 4.3 Shopping List App

It’s common for a person to quickly jot down a few items they require from the shops before going. In particular items required for a particular recipe. Write down some use-cases for a shopping-list application for a smartphone.

The user should be able to store recipes such that *all* the required ingredients are added to the current shopping list. They should in addition be able to remove some of those ingredients if they already have them.

Another good feature would be substitutions, such as ingredients which could be used in place of the desired ingredient/item if none are available.

Of course the user should be able to tick off items as they are placed into their basket or bought at particular shops. An implementation could make this the same operation as for removing items from a recipe that the user already has.

## 4.4 Synced Push Notifications

Many of you will have mobile devices such as smart phones and tablets. Many applications on such devices deploy push notifications. A background application which you are not using provides a notification that there is something which demands your attention.

A simple example is an email application which will notify the user when there are unread mail messages.

Some of you, may have *multiple* devices but wish to use the same services on your multiple devices. When a user touches a notification that action opens the corresponding application. That application may then remove the notification from the notification bar. It will not remove it from the notification bar of any other device that you own. Some action that you take (such as reading an unread mail) may mean that the background application on your other devices

removes it for you. Typically though, the notification remains, and it will certainly remain when the user does not actually open the corresponding application but merely swipes the notification to remove it from the notification bar.

Define some use cases for synced push notifications. Concentrate on the features that distinguish these from current push-notifications. Note that when defining use cases, you may wish to comment upon design *implications* but you do not wish to make any design decisions that are not enforced by the use-case.

#### 4.5 Basket Sub-total Calculation

Define use cases for the sub-total calculation for a customer's basket. This could be as used in a physical shop such as a supermarket or an online store. Assume that you have unique identification of the product kinds in the customer's basket associated with their price. However you must also apply any special offers. These include multiple-good special offers such as, buy-one get-one free, three-for-two, buy-one get-one half price. Note, that these offers often apply to multiple kinds of products, for example "buy any 3 for 2". Sometimes the product kinds do not even have the same price, usually it is the cheapest one which is free.

If you wish, as an extension, you can consider a special offers recommendation scheme. For example if a customer has 2 items which form part of a 3-for-2 offer, it could recommend additional items to be added for (little or) no cost.

#### 4.6 Twitter-bot

Write a use-case for a twitter-bot that warns users they have likely mis-spelt a popular hashtag.

#### 4.7 Semantic Highlighting

You are likely familiar with *syntax* highlighting, in which your source code is coloured according to the syntax of the programming language it is written in. Typically, keywords are one colour, operators another, and comments another.

Although this has some use, others have proposed a more helpful colouring termed *semantic colouring*. In this scheme, identifiers, such as variable names and types, are coloured rather than the syntax. Each name is given a new colour. So, for example the variable name  $x$  may be coloured some shade of blue, whilst the variable name  $y$  is coloured some shade of red. This makes it easier to the local uses of the same variable name. In particular matching a definition/assignment of a variable to its local uses.

Write a use-case for semantic highlighting for your "go to" programming language. To do this, you may wish to flesh out the details about what entities are coloured the same colour and perhaps from where new colours are derived.

### 5 Bad Use Cases

This section contains some examples of bad use cases for the above systems. Try to explain *why* each use-case is a bad one.

#### 5.1 Thermostat

**Use Case Name** Poll rooms

**Iteration** Draft

**Summary** The monitored rooms in the house should be polled for their current temperature.

**Basic Course of Events** Whenever the heating is being used the system needs to know whether to: turn the boiler on, keep the boiler on, turn the boiler off, keep the boiler off. To do so it needs to know if any of the rooms in the house are below the desired temperature, and to do so it must poll each of the monitored rooms.

**Alternative Paths** No obvious alternative paths.

**Exception Paths** One or more of the rooms' temperature sensors fails to respond. In this case we simply treat that room as an unmonitored room.

**Extension Points** No extension point.

**Trigger** Polling is done periodically for some set time interval.

**Assumptions** Assume that the rooms are all fitted with appropriate temperature sensors which can all respond to the poll request.

**Precondition** The heating is on and the user has set the desired temperature.

**Postcondition** The boiler is correctly on or off. That means that it is either off if all monitored rooms are at or above the desired temperature, or it is on.

**Author** Use case author.

**Date** Use case date.

## 5.2 Elevator

**Use Case Name** Efficient route

**Iteration** Draft

**Summary** When there are multiple calls for the elevator on separate floors the elevator should take the most efficient route to satisfy these requests.

**Basic Course of Events** For example, suppose there are 5 floors and the elevator has just stopped at the 4th floor arriving from below. There are calls for the elevator on the 5th floor and the ground floor. The elevator should first go to the 5th floor followed by the ground floor rather than vice-versa. Care should be taken however that any one passenger entering the elevator moves monotonically towards their destination floor.

**Alternative Paths** No alternative paths.

**Exception Paths** No exception paths.

**Extension Points** This is an extension of the basic use case allowing a user of the elevator to call the elevator.

**Trigger** There are multiple pending calls for the elevator.

**Assumptions** The user has indicated the correct direction in which they wish to travel within the elevator.

**Precondition** The same as the trigger.

**Postcondition** All calls for the elevator which were pending at the start have been satisfied.

**Author** Use case author.

**Date** Use case date.

### 5.3 Synced Push Notifications

**Use Case Name** Clear Push Notifications

**Iteration** Draft

**Summary** The user should be able to clear a notification without actually opening the application from which it has originated.

**Basic Course of Events** When the user swipes a notification there is a notification object instance of the Notification class. This calls its own `broadcastRemoval` method which notifies a `NotificationManager` object. This in turn, removes the notification from the notification manager *and* submits the removal request to a global notification sync service  
....

**Alternative Paths** The `NotificationManager` object is additionally responsible for opening the corresponding application should the user touch the notification. The `NotificationManager` can also remove the notification from the notification bar if the application requests it to.

**Exception Paths** An `QueueRemoval` class retains the notifications removal requests that cannot be sent immediately to the global notification sync service in the case that there is no current Internet connection available.

**Extension Points** This is an extension of simple push notifications that can be removed on the current device.

**Trigger** An application made a request for a notification to be added to the notification bar which the `NotificationManager` object responded to with success. The user has observed the notification but wishes to ignore the notification and remove it from the notification bar.

**Assumptions** The `NotificationManager` object is running and there is only one of them.

**Precondition** There is at least one notification in the notification bar and that the bar is in a state consistent with the `NotificationManager` object.

**Postcondition** The appropriate notification is removed from the notification bar which remains in a state consistent with the `NotificationManager`.

**Author** Use Case Author.

**Date** Use Case Date.

## 6 Advanced - Parameterised Use Cases

This section opens up a discussion for you to think about yourself. In programming languages we constantly evolve techniques to avoid writing the same thing more than once. This often involves parameterising some task, such as a method which takes a parameter. Consider doing the same for use-cases. Could we make effective use of some means of parameterising use-cases?