Inf2-SEPP 2020-21

Coursework 2

Creating a software design for a COVID-19 shielding food box delivery system

1 Introduction

The aim of this coursework is to create and document a design of the COVID-19 shielding food box delivery system, as well as reflect on some related Professional Issues concerns.

This coursework builds on Coursework 1 on requirements engineering, and needs to be attempted fully as a group. Please refer back to the Coursework 1 instructions for a system description. The follow-on Coursework 3 will deal with implementation and testing, and related professional issues.

This coursework is split into 2 practical parts:

1. A Software Engineering part (Section 3), including tasks and self-assessment.
2. A Professional Issues part (Section 4) including tasks and self-assessment.

1.1 High level self-assessment instructions

The self-assessment in each part will be part of your mark. For the self-assessment, you should consider to what extent your have met the assessment criteria for that part of the coursework. For each criterion, write one paragraph (a minimum of 2 sentences) discussing how well you think you have met that criterion and why (i.e., strengths and weaknesses of your solution). Moreover, we encourage you to reflect on things like: your experience with the approaches used, progress and work within your team, difficulties you encountered and how you addressed them, parts that you are particularly proud of and why, parts that you are not happy with and you could further improve, how you could improve them.

Important! You should make a real effort to be reflective (especially for the criteria where you have more to say), as well as honest, here. Please note that only making statements about the criteria without justifications (e.g., "We identified stakeholders excellently well") is not acceptable and will result in no credit for this part.

After the first deadline to this coursework, the markers will provide you with feedback on the same criteria to the one you have self-assessed on.

Moreover, we provide you with a set of ratings for each of the parts, which give you an understanding of your progress with that part of this coursework. For the Software Engineering part, we keep these ratings informal so that we can provide you with an initial idea of how well you did after the first (formative) deadline. Please note that these ratings do not have a direct correspondence with the grade intervals of the Extended Common Marking Scheme[1] which we will use for your final mark, but they are still meant to be informative. For the Professional Issues part, as the first deadline is the only and final deadline, the ratings already correspond with the grade intervals of the Extended Common Marking Scheme which we will use to compute your mark for this part.

2 Updates to the system description

The Scottish Government has been successful in establishing collaborations on the food box scheme with numerous catering companies spread across Scotland. A decision needed to be made as to which catering company would handle each pre-assembled food box order, such that as many orders can be delivered to shielding individuals as possible, and resources are not wasted. After discussions with the catering companies and the consideration of different options, a decision was reached that the catering company which has headquarters closest to a shielding individual's location (considering addresses, postcodes) will always be handling pre-assembled food box orders for that shielding individual.

As developers of the food order box system, you have raised the awareness of the Scottish Government about potential problems in allowing people to order from a supermarket system directly, as they may not be shielding individuals and thus eligible to get such free food boxes. You recommended, and your customer and the supermarkets agreed, that the food box order system should issue each user with a personalised single-use link allowing them access to a supermarket's dedicated food box order page. The supermarket system would check the link, as well as the request issuer's IP, to decide whether to offer the user access to this page in order to make an order.

[1] `https://web.inf.ed.ac.uk/infweb/student-services/ito/students/common-marking-scheme`

3 Your Software Engineering Tasks (worth 68.75% of the final mark for this coursework)

3.1 Design document

Ultimately, what you need to produce for this coursework is a design document that combines descriptive text with UML class, sequence and communication diagrams, describing the food box system. The following subsections specify what should be included in this document.

There is no requirement for you to use a particular tool to draw your UML diagrams. The *draw.io* tool[2] is one easy-to-use tool you might try. If you draw your diagrams by hand, be sure to include a high-quality scan in your report.

3.1.1 The system architecture

The food box system is intended to be used over the web, with shielding individuals potentially accessing it through different ways (we suggested SMS messages in CW1, but for the future the Scottish Government may also be considering web interfaces for it on computer, mobile and tablet). Moreover, the Scottish Government is foreseeing that they may need to extend the system in the future so that it is also used for individuals from remote areas in Scotland, and for the delivery of other goods like first aid kits and medication, or even the provision of emergency services. Therefore, the system may in the future need to interact with the systems of many other providers and organisations. It may also need to be used concurrently by a large number of users and systems.

Given your understanding of the system description, as well as your customer's plans for its future from above, what would be your choice of type of architecture for this system, from the ones that you studied in the lectures and required reading? Please justify your reply, making reference to what you know about the system and about your chosen type of architecture.

Important! For the rest of this assignment, you should NOT make any assumptions that any architecture (your proposed one or others) are used.

3.1.2 UML class model

Construct a UML class model for the system, also considering the updates to the system description mentioned in section 2. Include operations only when you consider them important for the execution of the following use cases:

1. **Register Supermarket**
2. **Register Shielding Individual**
3. **Edit Food Boxes**

[2] `http://draw.io`

4. **Place Order**

5. **Cancel Order**

Do not include simple operations such as attribute getters and setters. The operation descriptions must include types of any parameters and the type of the return value. Some methods may take or return collections of objects of a given class `T`; represent these as of type `Collection<T>` rather than specifying a concrete collection type, unless you already have the exact type in your diagram as a separate class. Associations must include multiplicities at each end. Leave navigation arrowheads off associations, as at this abstract level of design, this information is not needed.

To create the class diagram, you are encouraged to experiment with the class identification technique discussed in lectures, and/or the CRC card technique from the recommended reading.

You might find it useful to express the class model using more than one UML class diagram. For example, one diagram might show just class names and the associations and other dependencies such as generalisation dependencies between classes. Other diagrams might omit the associations, but show for each class its attributes and operations.

For this task, it is important to consider things which could help you make a good object oriented design, like design principles and design patterns.

You may find the following reference for drawing class diagrams in `draw.io` useful[3].

3.1.3 High-level description of the UML class model

The high-level description should expand upon the details of the design you chose. Focus on clarifying aspects which may be unclear from your class diagram and explaining where you have made design decisions. What alternatives did you consider and why did you make the choices you made?

You discussed ambiguities, subtleties, incompleteness in the system description during the first coursework. Be sure to indicate how these assumptions have impacted your design. A viewer of your class diagram should quickly form an impression of the structure of your design. But obviously the diagram leaves out many details. If you have attributes or operations of classes whose purpose is not clear from their names and associated types alone, add a few explanatory notes.

3.1.4 UML sequence diagram

Construct a UML sequence diagram for the *Edit Food Box* use case, focusing on the part of its scenarios which is about editing the food box after it was ordered. Show message names, but there is no need to include a representation of any message arguments. As needed, use the UML syntax for showing optional, alternative and iterative behaviour.

[3] `https://about.draw.io/uml-class-diagrams-in-draw-io/`

3.1.5 UML communication diagrams

While sequence diagrams focus more on the sequential exchange of messages between objects, the alternative notation of UML *communication diagrams* (also sometimes called *collaboration diagrams*) can be used to focus on the organisation of the objects and their interaction. Communication diagrams were introduced in Lecture 10 and its associated resources. You may find this link useful for constructing communication diagrams in *draw.io* [4].

Produce UML communication diagrams for the main success scenarios of the *Place Order* and *Cancel Order* use cases. For Place Order, please assume that the shielding individual does not decide to edit the food box as part of the scenarios.

3.1.6 Conformance to requirements

Your static and dynamic models should both be consistent with the system requirements, as captured in coursework 1. Before you tackle any of the software engineering tasks in this coursework, we recommend that you check marker feedback on coursework 1 and update your requirements and use cases accordingly in your coursework 1 solutions. Moreover, if you discover any issues with your requirements documents while working on this coursework, then you should also update them in coursework 1. Please highlight any changes you make to coursework 1 in green, there, before re-submitting coursework 1. In this section of coursework 2, you should comment on any apparent divergences with the requirements or issues you encountered with your requirements documents (particularly those which required changes).

3.2 The software development

Please provide short replies to each of the following questions:

1. Of the two types of software development processes that we have studied in this course, which of them is handling design more like in this coursework? Why do you think this is the case?

2. How does the other software development process handle design? How is this different to the first in terms of outcomes?

3.3 Software Engineering Task Self-assessment

Please read the high-level instructions for self-assessment from Section 1.1.

The following are the criteria you need to self-assess on regarding the Software Engineering tasks by writing one paragraph (at least two sentences) for each:

1. Reflection on the suitability of different types of software architecture in the context of the given case study, based on the system description/updates and the customer's plans for the future

[4] `https://about.draw.io/uml-communication-diagrams/`

2. Application of the UML class diagram notation to represent the static design of the food box system, including operations for the 5 given use cases, based on the system description/updates to it

3. Quality (from the point of view of good object oriented design) of the static design represented in the class model

4. Quality (in terms of correctness, effort) of the explanations and the justifications made for design choices in the high-level description of the class model

5. Application of the UML sequence diagram notation to represent the interactions involved in the Edit food box use case (scenarios of editing the food box after it was ordered), based on the system description

6. Quality (from the point of view of good object oriented design) of the interactions represented in the sequence diagram

7. Application of the UML communication diagram notation to represent the interactions involved in the main success scenarios of the Place Order and Cancel Order use cases, based on the system description/updates to it

8. Quality (from the point of view of good object oriented design) of the interactions represented in the communication diagrams

9. Effort at ensuring consistency between cw1 and cw2 solutions (as also reflected by your explanations in section 3.1.6).

10. Consistency within the cw2 solution (level to which your solutions to different tasks are aligned between them)

11. Reflection on how different software development processes handle design, in the context of the given case study

12. Effort at this self-assessment

For the formative assessment, you will receive a rating for the Software Engineering part among the ones provided below (this is only for your information, nothing to do here):

- **Worrying**: A solution where barely any or none of the SE concepts are correctly understood and applied to the case study. This feedback suggests that you should do a lot more work in studying the lectures, reading and practicing the tutorials, and then redo your solution to this coursework.

- **A lot to do**: A solution where only some of SE concepts are correctly understood and applied to the case study, and the resulting design has a poor level of quality (from the point of view of good object oriented design). This feedback suggests that you need to do more work in studying the lectures, reading and practicing the tutorials, and then re-consider your solution.

- **Getting there**: A solution where a good amount of SE concepts are correctly understood and applied to the case study, and the resulting design showcases a reasonable level of quality (from the point of view of good object oriented design). Additionally, the solution is mostly consistent within and with CW1, and the self-assessment is honest but only focuses on strengths and weaknesses.

- **Looking good!**: A solution where all SE concepts are correctly understood and applied to the case study, and the resulting design showcases a good level of quality (from the point of view of good object oriented design). Moreover, the solution is consistent within and with CW1 and the self-assessment is thorough and reflective, going beyond the discussion of strengths and weaknesses.

3.4 Further information

3.4.1 Modelling using message bursts

Create a system design with a single thread of control. Do *not* try to make it concurrent.

System activity consists of bursts of messages passed between objects, each triggered initially by some actor instance sending a message to some system object. Each burst corresponds to the execution of some fragment of a scenario of a use case. Assume that the only messages sent from system objects back to actor instances are the final reply messages of bursts. Such a message corresponds to the return of the method invoked by the trigger message sent by the actor. Imagine each burst completes relatively quickly, in well under a second.

Because of the single-threaded nature, the system does not handle further input messages during a burst. This should not be too much of a restriction, because of the assumed short duration of each burst.

3.4.2 Abstract inputs

Consider input messages at an abstract level. Do not model the user interface or different sources of the input messages. Here are some examples: Here are some examples:

1. To register, a *shielding individual* would send in a *register shielding individual* message, providing their CHI number, which is then checked against the Public Health Scotland records to verify that the individual is indeed shielding.

2. To get a food box, a *registered shielding individual* could send a *get food box* message, opting for either catering company or supermarket.

3. If going for the catering company, the user can optionally send an *edit food box* message with the food box identifier and the information about the edited items and their quantities, and then a *place order* message, providing the delivery date and the edited food box or all of its contents, to place the order. The system then returns the placed order id.

3.4.3 Abstract data

Your design should create a Location class which is used to specify the details of a location (which consists of an address and a postcode). This will need to include a method for testing whether two locations are near to each other — do not worry about how this should be implemented yet.

Your design will also need to handle dates (useful for the delivery of the food box). Dates should be represented using Java's `LocalDate` class[5].

3.4.4 Abstract outputs

Do not model the complexities of the user interface that might be displaying results. You should focus on designing the software system.

For example, a food box can be returned as an object with appropriate fields containing its details, and some of the messages can return a string (e.g., the status of the order or the verification of a command being successful).

3.4.5 Catering Company and Supermarket Systems

Although the catering company system is very important in the processing of food box orders, it is an external system and so you must not concern yourselves with how it works. However, make sure that your design allows our system to interface with the catering company system for checking its stock levels, notifying it of food box orders being made, and getting status updates.

Supermarket food box orders are completely handled on the supermarket system, by providing users with a personalised link to them as explained in Section 2. However, our system must also provide a way for the supermarket to notify it of an order being placed, and subsequent order status updates. Add a way for the supermarket system to interface with our system in your design.

3.4.6 Delivery Service System

Delivery services may be employed by the catering companies to deliver food boxes to the locations of shielding individuals. You do not need to be concerned with the details of how this works. Delivery services only need to notify our system of the status of food box order delivery, and so you should make it possible for a delivery service system to interface with our system this way in your design.

[5] `https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html`

4 Your Professional Issues Tasks (worth 31.25% of the final mark for this coursework)

As a team, write a short essay (we will refer to them as 'blog posts') on each of the topics below.

Each blog post should be a maximum of 500 words (this amounts to about one side of A4), and good submissions will not be far under this.

This article gives good advice on how to structure a short essay:

`https://www.wikihow.com/Write-a-Short-Essay`

Remember that reading the first chapter of "A rulebook for arguments" should help here too.

Good essays will:

- Clearly choose a main position
- Clarify important details with reference to other sources
- Consider the terms used and define them where necessary
- Justify arguments with reference to course materials
- Anticipate and address counterarguments

How well you do these things will be the core criteria for marking in your blog posts.

4.1 Topics

Topic 1: "Discuss the extent to which the Volkswagen emissions testing scandal was a failure of Organisational Structure and Management."

The case study is available here: `https://www.bbc.co.uk/news/business-34324772`

This topic is an opportunity to reflect on an event that has already happened, and so lends itself to exploring other sources, as well as reading and referencing others' thoughts on the topic.

Topic 2: "How might legal considerations around Intellectual Property impact development of your food box delivery system?"

For the purposes of this question, you can assume that you are actually developing the system outside of a University course, as a company. It is also encouraged to imagine how the system might be expanded further, both technically and as a product (for example being bought by another company). This question less obviously invites use of external sources, but reference to other cases as examples is still a good way to strengthen an answer.

4.2 Professional Issues Self-assessment

Please read the high-level instructions for self-assessment from Section 1.1.

The following are the criteria you need to self-assess on regarding the Professional Issues blogs by writing one paragraph (at least two sentences) for each (they are the same as for CW1):

1. Answers the question
2. Clarity
3. Appropriate structure
4. Supported by other sources
5. Quality of argument
6. Recognition of counterarguments / alternate views
7. Knowledge and understanding
8. Style and presentation
9. Effort at this self-assessment

For the only assessment for these tasks (final one), you will receive a mark according to the following marking scheme and respecting the Extended Common Marking Scheme (this is the same as for CW1, and only for your information, nothing to do here):

- Pass (40+): Essay attempts to address the provided question but is hard to understand and/or makes few clear points.
- Good (50+): Essay is understandable, clarifies key terms, and comes to one or more obvious conclusions. Some course materials and external sources are referenced.
- Very good (60+): Essay has a good structure, flowing between and building upon subsequent points. It identifies possible counter arguments or alternative views and integrates a variety of external sources. The self-assessment is honest, but not very reflective.
- Excellent (70+): Essay reads well and contains well-made arguments which pull together a variety of views and sources. The self-assessment shows extra effort in reflection.
- Excellent (80+): Marks in this range are uncommon. This essay draws the reader in and makes points beyond what would be expected of undergraduate students in Informatics.

5 Some advice (same as for CW1)

5.1 Working online as a team

Teamwork is not easy, and this year it is made harder by the fact that we work remotely. However, you can turn this to your advantage if you use your experience as Informatics students, and make use of the wealth of software tools available to help you. Here are some that we would recommend:

1. Microsoft Teams (free through our university) for setting up a team with your colleague, setting up meetings in the calendar, video calls, the chat, editable file sharing, its Tasks By Planner and To Do to organise and split your work.

2. OneDrive or SharePoint under Office365 (free through our university) for storing documents, sharing and working collaboratively on them.

3. The GitHub (`https://github.com/`) online repository and version control system, which you will also use later in this course. Please be careful about access permissions (see subsection 6.3).

4. Trello (`https://trello.com/en`) as an excellent (and also free) alternative for splitting up work and recording progress on tasks. We will also use it later in our course.

5. Aww (`https://awwapp.com/`) as an online whiteboard where you can collaboratively sketch your ideas like you would do on paper.

You may want to mention what tools you used for your teamwork in your self-assessments.

5.2 Asking questions

Please ask questions in labs, office hours or on the class discussion forum if you are unclear about any aspect of the system description or about what exactly you need to do. On the class discussion forum, tag your questions using the *cw2* folder for this coursework. As questions and answers build up on the forum, remember to check over the existing questions first: maybe your question has already been answered!

5.3 Good Scholarly Practice

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

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

Please note that we will run a plagiarism checker on your solutions.

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on an online repository like GitHub then you must set access permissions appropriately (permitting access only to yourself or your group).

6 Submission

6.1 For the Software Engineering Tasks in Section 3

Please submit a PDF (not a Word or OpenOffice document) of your design document. The document should be named **design.pdf** and should include **a title page with names and UUNs of the team members who have worked on this coursework**.

As a group, *only one of you* needs to make the submission. If you submit several times, only the last submission will be checked by the markers.

How to Submit

Ensure you are logged into MyEd. Access the Learn page for the Inf2-SEPP course and go to "Coursework" – "SE Coursework Submission" – "Coursework 2".

Submission is a two-step process: upload the file, and then submit. This will submit the assignment and receipt will appear at the top of the screen meaning the submission has been successful. The unique id number which acts as proof of the submission will also be emailed to you. **Please check your email to ensure you have received confirmation of your submission**.

If you do have a problem submitting your assignment try these troubleshooting steps:

- If it will not upload, try logging out of Learn / MyEd completely and closing your browser. If possible, try using a different browser.
- If you do not receive the expected confirmation of submission, try submitting again.
- If you cannot resubmit, contact the course organiser at Cristina.Alexandru@ed.ac.uk attaching your assignment, and if possible a screenshot of any error message which you may have.
- If you have a technical problem, contact the IS helpline (is.helpline@ed.ac.uk). Note the course name, type of computer, browser and connection you are using, and where possible take a screenshot of any error message you have.
- Always allow yourself time to contact helpline / your tutors if you have a problem submitting your assignment.

6.2 For the Professional Issues Tasks in Section 4

Please submit your blog posts for the markers to view as a PDF in 'Coursework" - "PI Coursework Submission" - 'Group Blog for CW2"

7 Deadlines

The deadlines for this coursework are as follows:

- **Deadline 1 (compulsory with final marks for the Professional Issues part!, formative for the Software Engineering part):**
  16:00, Mon 1st March

  Please submit the Professional Issues part of this coursework for final marks at this deadline. Please submit the Software Engineering part of this coursework for formative marking at this deadline, to get formative feedback back.

- **Deadline 2: (compulsory with final marks for the Software Engineering part!)**
  16:00, Fri 9th April

  For this final deadline, please consider the feedback received from the markers on the Software Engineering part of this coursework after the first deadline and re-submit this part, highlighting your changes to your solutions in green. Also, adjust your self-assessment justifications to explain how you have addressed marker comments, also as highlighted in green.

This coursework is worth 30% of the total coursework mark. The mark for this coursework consists of the mark for the Software Engineering tasks (worth 68.75%) and that for the Professional Issues tasks (worth 31.25%). This mark will become final after deadline 2 mentioned above.

Boris Mocialov, Chris Vasiladiotis, James Garforth, Cristina Alexandru, 2021.