Inf2-SEPP 2022-23

Coursework 3

Software implementation of a customisable events app

This coursework requires teamwork for SE (1), and individual work for ProP (2).

1 Your SE work (team work)

The aim of this coursework is to implement and test the customisable events app which facilitates the management and attendance of social events for AcmeCorp. It builds on Coursework 1 on requirements engineering and Coursework 2 on design.

The following are the documents that you will primarily need:

- CW1 instructions and CW2- New or Changed Requirements Document
- Development Tools document
- CW2 solutions with CW2 solution sequence diagrams
- CW3 starter code
- CW3-SE high level marking scheme

Please never refer to your CW1 or CW2 solutions in this coursework.

1.1 Requirements and their updates

In this coursework, you should consider all the requirements that can be derived from CW1- Section 1 and subsections, CW2- New or Changed Requirements document (see 1). Moreover, your company has recently received clarifications on some of the stakeholders' needs, and so it has come up with the following decisions, which you should also consider:

- Tags shall be represented as name-value pairs, with a default value provided. The system shall include default tags for social distancing, air filtration and venue capacity. When creating a new tag, staff can set its name, possible values it could take and the default value. They can select the value for each tag when creating an event. The system will need to check that the provided names and values will only include known names and values. On setting up their profile, consumers can select the values for all tags that had been used by staff throughout their events.
- Addresses shall be represented as latitude-longitude, and checks for address formats will always involve checking that the are valid latitude-longitude coordinates which additionally fall within map system boundaries. Consumer addresses shall be validated when the consumers register, as well as when they update their profiles (which may involve updating their address too). Venue addresses shall be validated on event creation by staff members (i.e. when they are provided).
- The following will cause the whole loading (i.e. importing) operation to be cancelled: for event tags, clashing tags (with the same name but different values), as this would affect events and consumers using these tags; any user email clashes for users that are not the same; for events, clashing events (with the same title, start date and tiime, and end date and time) that are not identical because otherwise this could result in either duplicated events or issues with connected bookings; for bookings, clashing bookings (by the same consumer for the same event with the same booking date and time) because otherwise this could result in duplicated bookings. Moreover, the current user is not replaced. If there are no clashes, identifiers (like event numbers or booking numbers) are renumbered for the newly loaded items to prevent conflicts.
- Regarding how events with unlimited tickets should be handled by the system - the system does not need to support this; staff can just enter a very high ticket cap to not enforce one.
- Regarding how free events should be handled by the system - staff can set the ticket price to 0. The system should have special handling for this case and not issue 0-value transactions via the payment system.

1.2 Design documents, updates to the old code

After further refinements during the design stage, your team has come up with further documentation that describes the new system. This is provided to you in the form of the CW2 solutions (see 1). Specifically, the solutions include a full class diagram (Section 3.1) and associated explanations (same and Section 2 point 3) for the entire new system, and sequence diagrams for some specific use cases (the ones discussed in CW2). ***IMPORTANT! The code you implement for this coursework should precisely follow the class and sequence diagrams in the CW2 solutions, for the use cases that you are required to cover for your team size***.

In addition, your company still has access to the old system's code. But even better than that, a colleague in your team has updated that code to include some basic changes for the new system. Specifically, the changes are: the code works only for one entertainment provider and its members of staff; an "organisation secret" String has been incorporated, and staff need to match it when registering an account to address security concerns; there is functionality for listing all bookings for a certain event for staff; there is functionality for listing all bookings for a certain consumer for that consumer. This modified version of the old code has been given to you as starter code (see 1). ***IMPORTANT! You are required to start implementation from this starter code in your solutions.***

1.3 Using tools, agile approaches and practices

This coursework is a small-scale opportunity to try out approaches, practices and software tools that have been mentioned in the lectures starting with Week 6. ***For this coursework (only), please assume that you are using an agile software development process***. As a result:

- You should pay attention to the *agile principles* from Lecture 2
- You are encouraged to use any applicable *practices* that have been described in Weeks 7, 8 and 9 of the course and that have been proposed by one of the following *agile approaches*: Extreme Programming, Scrum, Kanban. Here, you do not need to stick to one approach, but you are free to mix and match practices from different approaches that would make sense in your view for the system that you are developing and your team. Please be careful about practices that are dependent on one another!
- You are encouraged to try out software tools that are recommended for your chosen approaches and practices, but also experiment with other tools that may be helpful for construction, testing and teamwork. Overall, apart from the required tools mentioned in the next section, you could try out for example GitHub and the Git version control system [1], a test coverage tool like the default one provided by IntelliJ IDEA[2] or one of its alternatives, a bug tracking tool like Trac[3] or JIRA[4], a static analysis tool like SpotBugs[5] or Infer[6], any tools for support with agile like JIRA, and any tools for team work as presented in the teamwork resources.

It is up to you how many and which tools, agile approaches and agile practices you use, and it should be based on your judgement of what could be beneficial for developing this system. Some of these may prove to be real life savers! The effort you put in will also

[1]see the Git and GitHub tutorial from the Learn course under Other Resources

[2]`https://www.youtube.com/watch?v=QDFI19lj4OM`

[3]`https://trac.edgewall.org/`

[4]`https://www.atlassian.com/software/jira/bug-tracking`

[5]`https://spotbugs.github.io`

[6]`https://fbinfer.com`

influence the quality of the reflection which you make in Task 6 and your assessment for this section.

1.4 Required Development Tools

AcmeCorp requires its developers to use certain development tools. We provide a Development Tools document that specifies these tools, their required versions, and how to install them (see 1).

You will also need to know how to create and run tests using JUnit (version 5), how to use interfaces in Java. If you are not comfortable with either of these topics you should review resources online. We suggest the following links: `https://www.vogella.com/tutorials/JUnit/article.html`, `https://bit.ly/3clJLs6`, `https://bit.ly/38t1ltn`, `https://www.w3schools.com/java/java_interface.asp`

1.5 Following good practice in construction and testing

Throughout this coursework, follow good coding practices as described in the lecture. You should attempt to follow the coding guidelines from Google at `https://google.github.io/styleguide/javaguide.html`

A common recommendation is that you never use tab characters for indentation and you restrict line lengths to 80 or 100 characters. You are strongly recommended to adopt both of these recommendations because it is good practice and, particularly, in order to ensure maximum legibility of your code to the markers. A good practice is to adopt a consistent formatting style either via your IDE or a standalone tool (which can be used via an IDE) such as `http://astyle.sourceforge.net/`

Be sure you are familiar with common interfaces in the Java collections framework such as `List`, `Set`, `Map`, and `Queue/Deque` and common implementations of these such as `ArrayList`, `LinkedList`, `HashSet`, `TreeMap`. Effective use of appropriate collection classes will help keep your implementation compact, straightforward and easy to maintain.

Browse the guidance at `https://www.oracle.com/technetwork/java/javase/tech/index-137868.html` on how to write Javadoc comments. Follow the guidance on including a summary sentence at the start of each comment separated by a blank line from the rest of the comment. This summary is used alone in parts of the documentation generated by Javadoc tools.

For testing the system using system tests, you are encouraged to adopt a test first approach (see Lecture 18). Make use of the provided tests to help you develop your code, and when tackling some feature not already tested for, write a test that exercises it before writing the code itself.

Sometimes it is seen as important to have development teams and testing teams distinct. This way there are two independent sets of eyes interpreting the requirements, and problems found during testing can highlight ambiguities in the requirements that need to be resolved. As you work through adding features to your design, you could alternate who writes the tests and who does the coding.

Use JUnit 5 to your advantage by using the right methods in your test class, the right annotations to test methods, and the right assertion methods (see Lecture 19).

Your system test files will serve as your primary evidence and documentation for the correct functioning of your system. In terms of their documentation, take care with how you structure your test methods, give them intuitive names, add appropriate comments (for example noting particular features being checked) and include a clear message when each test fails.

1.6 Your Tasks

There are several concurrent tasks you need to engage in. These are described in the following subsections.

Bear in mind that it is not enough to submit code claiming to implement your system without testing and other documentation demonstrating its effectiveness. As part of this assignment you are asked to implement tests demonstrating the correctness of some classes, and how your system implements the key use cases of the system, and **these tests will be used as the primary means of marking this coursework**.

1.6.1 Task 1: Justifying decisions

This consists of two sub tasks.

1. **Task 1A**: The AcmeCorp developer teams have been deliberating which map API to use. Someone proposed GraphHopper, and a decision was made to use it. Look into the documentation for this API, having in mind the main uses you want to get from it for your use cases (mentioned per team size in Task 2 below). Then, also research one other Java library that could be used for implementing the map system for your uses. What would be its main 3 pros and 3 cons compared to GraphHopper in your opinion? Please provide some links or references to support your statements.

2. **Task 1B (Teams of 3+)**: A fellow developer has proposed that you use serialisation of the main objects holding data about users, events, bookings and used event tags. This can help address the risk of losing all the system's data if the kiosk machine crashes. Moreover, serialisation can be used for saving/exporting data, and deserialisation (its reverse) for loading system state/importing data. You can start reading about serialisation here. One option for doing serialisation/deserialisation is using the Java Serialisable API, which can help save and restore objects in a binary data format. You can read about it here. The alternative is to save all the information about the objects using JSON, XML, YAML, or another human-readable data format. Research these different alternatives. Then, indicate your

agreement/disagreement with the following statements (i.e. whether each of them is true or false), providing *at least one justification* for each, and links/references to support your justification(s):

(a) An advantage of using binary formats like the Serialisable Java API versus using a human-readable data format is that data is quicker to parse by the machine. **(Teams of 3+)**

(b) Data serialised using the Serialisable API can be loaded partially. **(Teams of 3+)**

(c) Data stored using a binary format like the Serialisable API takes more disk space than data stored using human-readable formats. **(Teams of 4)**

(d) Data serialised using a human-readable data format can be edited on the fly using a text editor. **(Teams of 4)**

1.6.2 Task 2: Construct code

Using the starter code (see 1), construct the code that implements the following use cases which are not already correctly handled by it (Note: these are slightly different than in CW2):

1. **Add Event Tags**
2. **Create Event**
3. **Register (Consumer)**
4. **Edit Profile (Consumer)**
5. **List Events (Consumer)**
6. **List Events (Staff)**
7. **Book Event**
8. **Request Directions to Venue**
9. **List Events by Distance**(only for **teams of 3+**)
10. **Review Event** (only for **teams of 3+**)
11. **List reviews**(only for **teams of 3+**)
12. **Export (i.e. Save) Data** (only for **teams of 4**)
13. **Import (i.e. Load) Data** (only for **teams of 4**)

Some of the above use cases require modifications (sometimes small) to the starter code, others require completely new functionality. We expect you to reuse as much of the starter code as possible, and not to modify its underlying structure. However, if you have strong reasons to do otherwise, you should justify your choices (also with regards to the design diagrams, see below) in the text.

Make sure you closely follow the relevant parts of the class model and the sequence diagrams (see 1). Your code should ideally perfectly match the provided class and sequence diagrams. However, if you find that something from them cannot work in your implementation, find a solution and justify in the text why you could not respect the design models.

Make sure to add comments (both JavaDoc and inline) to your methods. You should make sure to explain what the various parts of your code are doing, such that a colleague (or a marker) can understand it easily.

Include inline assertions in at least some of your methods (minimum two) as a means of catching faults and invalid states, but please make sure that you turn these off before submitting.

Advice for writing high quality code was provided in section 1.5.

Unlike CW1 and CW2, this coursework doesn't have a dedicated "ambiguities and assumptions" task. However, ***you should still make sure to mention any assumptions you make in this task***.

Reaching an "Excellent" mark in this task

If you become confident that your solution to this task correctly respects all of its instructions from above and is of high quality, and you want to be able to reach a mark of "Excellent", you need to put in effort to go beyond what was required to impress the marker. The following are some possible ways:

- Putting a lot of effort into checking for faults, invalid states, or doing input validation to the code
- Looking for "bad smells" in the code and proposing ways that the code could be refactored (in writing only, without modifying the code which will be checked for conformance with the class and sequence diagrams). Here, please include small code excerpts in the report as needed to support the explanation. See Lecture 17.
- Considering non-functional requirement (i.e. quality attribute) categories and proposing ways that the code could be improved to improve them (in writing only, without modifying the code which will be checked for conformance with the class and sequence diagrams). As above, please include small code excerpts in the report as needed to support the explanation.

This, together with excellent tests, can lead to exceptionality marks too.

1.6.3 Task 3: Create system-level tests

Use JUnit 5 to create system-level tests that simulate the scenarios of each of the use cases you implemented in the last task (still respecting group sizes). Some examples of system tests are already provided for you in the starter code.

To get a good mark for this coursework you must test all the use cases you implemented, with several scenarios each, and make sure that they all pass, since your tests will provide the main means for assessing your implementation. Even if you have implemented a feature, you may not get full marks if you have not tested your system sufficiently to demonstrate your system successfully implements the feature.

Few tests are able to test single use cases by themselves. In nearly all cases, several use cases are needed to set up some state, followed by one or more to observe the state. For example, you you will need to register a consumer, register an entertainment provider and for it to create an event, before the consumer can book that event. Additionally, you may want to include tests which perform more than one use cases in turn to check the integration of the modules of the system.

Do not run all your tests together in one large single test. Where possible, check distinct features in distinct tests. However, don't rely on the order of the tests and Java does not necessarily run them in that order.

Include all your system tests for each use case as JUnit test methods in a `[use-case-name] SystemTests.java` class (e.g. `AddEventTagsSystemTests.java`, `CreateEventSystemTests.java` etc.). Overall, you should have one such class for each use case.

Other advice advice for testing was provided in section 1.5.

Reaching an "Excellent" mark in this task

If you become confident that your solution to this task correctly respects all of its instructions from above and is of high quality, and you want to be able to reach a mark of "Excellent", you can try to cover through your system tests most if not all of the different scenarios for the each case that you are tackling. This, together with excellent code, can lead to exceptionality marks too.

1.6.4 Task 4: Unit testing

In addition to the system tests from the previous task, please provide unit tests for the following classes:

1. `Consumer`
2. `Booking`
3. `Event` (only for **teams of 3+**)
4. `EventTagCollection` (only for **teams of 4**)

These should focus on checking whether the implementation of each specific class is correct, and all pass. Place the tests for a class named `MyClass` in a file named `TestMyClass.java`, and use JUnit 5 assertions and annotations similarly to the examples provided in lectures and Tutorial 6.

You don't need to test classic getters/setters, only any that do more than usual.

In each test class, you should have several test methods for each method of the class that you are testing. Each test method should test for one combination of inputs as parameters. Make sure that you include frequent classes of inputs, but also more unusual ones, boundary and incorrect inputs for testing (see Tutorial 6 and its solutions for help). Each test method should be appropriately named such that the case you are testing is clear. Each test method should run the method that is being tested only once, i.e. you would normally have one single `assert` statement in each test method. Finally, include a clear message when the test fails.

Other advice advice for testing was provided in section 1.5.

Reaching an "Excellent" mark in this task

If you become confident that your solution to this task correctly respects all of its instructions from above and is of high quality, and you want to be able to reach a mark of "Excellent", you can try to test for all classes of inputs for each method and also use a tool to calculate statement and branch coverage in each of the tested classes and improve your code and retest until they are at least 70% (see Lecture 19). This, together with excellent code, can lead to exceptionality marks too.

1.6.5 Task 5: Code review (optional, for bonus marks)

This task is optional, but a great opportunity to get some feedback on a small part of your code - specifically, your AddEventTagCommand class implementation and its system tests - and exercise doing a code review which is a useful skill for a software engineer. Moreover, it can result in 2% bonus marks.

If interested, please join one of the Week 10 or 11 labs as a team, having prepared in advance the AddEventTagCommand class implementation. There, one of the lab demonstrators will first ensure that you have the necessary code (precondition for the code review). Then, the demonstrator will pair you up with another team and you will be asked to exchange your implementation with them (and only them), and work separately to assess each other's work on them and write your notes in a text file, for 20 minutes. At the end of your allocated time, you will be asked to exchange your notes with the other team, and also submit them in an area on Learn which will be dedicated to the code review, all under the supervision of the demonstrator.

You will be provided with some guidance on what constitutes a good code review, and where you can submit your code review on Learn, in due course through a Learn announcement.

1.6.6 Task 6: Reflection (same as in CW2 apart from size limits, subtask 3 (marked as NEW!) and the marking scheme)

This section asks you to reflect and self-assess your team's progress with this coursework. We think such reflection could have an impact on your learning and of your understanding/expectations of how you will be marked.

Important! You should make a real effort to be reflective, as well as honest, in this task. Please note that only making bold statements like "We did this excellently well", with no justification, and (even worse!) not being open to consider that there is always room for improvement, will result in very little, or even no, credit for this part.

Start by having a look at the following reflection model (adapted from the Integrated Reflective Cycle (Bassot, 2013)) that you are asked to use to structure your reflection:

1. **The Experience**: Describe what you did, what you tried out.
2. **Reflection on Action**: What were the results? What went well? What didn't? Why?
3. **Theory**: What have you learned from this experience?
4. **Preparation**: What could you have done to make things better, according to the lessons learned? If you have the chance to do this again (e.g. team work), what will you do or try out next time to try to make things better?

You can see an example of this model being put to use at this link.

1) Reflection on teamwork

Using the above reflection model, write one- two paragraphs summing up to *maximum 250 words* of reflection on your team's teamwork for this coursework. Focus on things such as how you got organised, split up responsibilities between team members, communicated, and managed progress in working towards the deadline for this coursework. Make sure to mention and reflect on the use of and usefulness of any tools that you tried out in this process, e.g. physical or online tools for managing your team work. You can use the reflection model repeatedly to describe how you improved your teamwork over time.

2) Reflection on the quality of your work (can lead to exceptionality marks)

Using the same reflection model, write one or two paragraphs summing up to *maximum 250 words* of reflection on the quality of your work. Mention how well you think you tackled the work in the different tasks. We recommend you have a look at the marking scheme from this link - making specific reference to parts of it- to help you structure this response, however touching on all marking criteria or marking yourself using it is not expected. You can use the reflection model repeatedly to describe how you improved your solutions over time.

3) NEW! Reflection on tools, agile approaches and practices (can lead to exceptionality marks)

Using the same reflection model, write one or two paragraphs summing up to *maximum 350 words* of reflection on your use of tools, agile approaches and practices for this coursework (see 1.3). In particular, here it is useful to reflect on your reasoning for choosing each approach/tool/practice (and not choosing an alternative), how the approach/tool/practice worked for you, what you would conclude are its advantages/disadvantages, what you would do different next time you develop a system. ***IMPORTANT! In your description, please include some screenshots of your use of any tools that you mention***.

1.6.7 Declaration of work

You are also required to declare the amount of work that was carried out by each of your team members, so that we can adjust your individual mark accordingly. In particular, you are each expected to split up work *fairly*. This means each team member doing:

- **Teams of 2**: around half (50%) of the work
- **Teams of 3**: around a third (33%) of the work
- **Teams of 4**: around a quarter (25%) of the work

on this assignment (no matter how you split responsibilities), or to have agreed with teammates how to average out across courseworks in case of personal circumstances.

For this task, prepare a separate document (see submission details in Section 4) and do either a) or b) from below:

a) If you consider *your work was split fairly according to the definition from above*, include in it the text ”The work was split fairly between our team members”, and all sign (ideally, see Terms below).

b) If not, write for each team member an estimate (in percentages) of how much work they have carried out. ***IMPORTANT! This estimate should ideally be agreed with the other team members, who should all sign this document (see Terms below)***.

At the end of the document, each team member should include a digital signature or simply a picture of their signature taken with their mobile phone.

Terms: Failure to submit this document or submitting it after the deadline + 15 minutes will lead to our assumption that you have split the work fairly, and you will all receive the same mark without any objections possible. We will accept a document without all signatures included, as long as it is submitted before the deadline + 15 minutes. However, note that submitting it without all signatures will lead to potentially difficult discussions between the course organiser and the team members. Finally, option a) done correctly will result in the same mark for all the team members, while option b) will mean bringing the mark down for anybody with a lower percentage than expected, in accordance with the declared percentage (e.g. if a team member in a 4-person team did

15% of the work, this is 60% of the 25% they needed to do, so they will receive 60% of the team's mark). Individuals who have done more than their share will not have their mark increased according to the percentage, but may respect criteria for excellence and exceptionality for a very high mark (see marking scheme).

2 Your ProP Work (individual)

As part of CW3 (ProP) you will be required to write an essay ***choosing ONE of the topics below***.

The length of the essay should be around 750 words and maximum 1000 (excluding the "optional extra" parts which should be maximum 250 words).

2.1 Topics

Topic 1: "Discuss different organizational forms of companies and their implication on agile projects and software engineering in general. As an optional extra describe a hypothetical optimal organization form of a company for agile projects"

Companies and their various organizational forms have a direct impact on agile projects and the way software engineering is performed. They can support or hinder, enable or hobble or anything in between.

Review some organizational forms and analyse (by research) how these interactions are and what is good and bad. Create your own thoughts what might be beneficial, what less; yet don't just focus on the agile part but the wider software engineering as well as this is impacted by organizations as well, or isn't they?

You are not expected to cover every aspect, yet a good overview of the organizational forms (and hybrid ones) and some impacts (if any) would be good.

Have a thought about Conway's Law in this context as well (see this link) – might be helpful when it comes to the "optional extra" question, which is like a logical follow up reasoning based on the main topic.

Topic 2: "Discuss the impact of one software engineering standard of your choice for the project implementation and which meaning this has for a production of high-quality code. As an optional extra briefly describe how a software engineering standard which address functional safety would impact your software engineering process"

There are many software engineering standards and you should reflect how one of these (pick your favourite and explain a bit why) interacts / interferes with your implementation. High-Quality-Code is dependent on many things – how does your standard of choice interact / interfere? Does it not at all?

These links might help you a bit and provide a basic reading and basis for further exploration: link 1, link 2.

2.2 Writing the essay

When writing the essay, you should think about the assessment criteria for this coursework – see the next section. Read carefully the criteria definitions and use them to guide your writing.

Here is a possible approach for writing your essay:

1. **Revisit relevant (course) materials**. Revisit lecture 1, 3, 8 and 17, and other useful sources (e.g., the first chapter of "A rulebook for arguments"). Carefully read the coursework description, particularly section 4 Assessment. Also, you may want to use this article as a guide for writing your essay.
2. **Brainstorm**. Take some time to write down your premises, conclusion, potential evidence. Write down ideas that support your conclusion but also opposing views. Think about various perspectives and write down ideas. Do not care about your writing style or structure/organisation of ideas, just write them down, preferably using bullet points.
3. **Prepare**. Collect all the evidence and write down an outline of your essay. Take enough time to gather the evidence you need and check if it was reliable. This is the phase when you organise your ideas written down in phase 2 and think about how to unfold them (e.g., in which order).
4. **Draft**. Write a rough draft of your essay. Don't spend time checking how correct your sentences are. Just write and try to include any data/direct quotes as early as possible.
5. **Revise**. Polish your rough draft, optimise word choice, make sure your sentences are concise, and restructure your arguments if necessary. Make sure your language is clear, and double-check that you effectively made all your points and rebuttals. Also, remove redundant sentences and sentences which do not bring any value to your argument. Do not expect to write your essay in one go, but develop it iteratively.
6. **Proofread**. Read through your essay and focus exclusively on fixing mistakes. Use Grammarly to identify and fix grammatical errors.

2.3 Assessment for the ProP Tasks

Please find here a marking scheme for ProP.

3 Some advice (Same as for CW1 and CW2)

3.1 Working as a team (for SE tasks only)

To get organised and work effectively as a team, you may want to have a look at the Teamwork Resources, and follow any advice on teamwork from guest lectures. You may also want to mention what you have done and used for teamwork in Task 6.

3.2 Asking questions

Please ask questions in labs or on Piazza if you are unclear about any aspect of the system description or what tasks. On Piazza, tag your questions using the *cw3* 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!

3.3 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.

You are also required to take reasonable measures to protect your assessed work from unauthorised access. E.g., if you put any such work on an online repository like GitHub then you must set access permissions to allow access only to you and your team.

4 Submission

Please make **three** submissions for this coursework (**IMPORTANT!** Some are group submissions while for ProP it's individual):

- **As a group**:
  1. SE task solutions, under "Assessment" - "Coursework Material and Submission" - "Coursework 3"- "Software Engineering Tasks"- "Submit Report". This should include:
     - A .ZIP file of your implementation, including all your tests and Javadoc as well. To do this in IntelliJ, go to:

       `File - Export - Project to ZIP file`.

       Rename this file **CW3SEImplementationGroupX.zip**.
     - A PDF (not a Word or OpenOffice document) of your report. The document should be named **reportTeamX.pdf**, where you replace the X

with your team number. Please **do NOT include the names and/or UUNs of the team members** within this document, so that we can mark anonymously. Only one of the team members needs to submit this document, and the last submission will be considered for marking if several team members submit.

2. The SE team declaration of work, under "Assessment" - "Coursework Material and Submission" - "Coursework 3"- "Software Engineering Tasks"- "Submit Declaration of Teamwork". This should include a PDF (not a Word or OpenOffice document) of your 'Declaration of work', entitled **CW3SEDeclaration GroupX.pdf**, where you replace the X with your group number. This document will only be accessed by your course organiser. This submission should also only be made by one of the team members, but this team member can be different to the one who submitted the requirements document.

- **Individually:** Your individual ProP report, under "Assessment" - "Coursework Material and Submission" - "Coursework 3"- "Professional Practice". This should include a PDF (not a Word or OpenOffice document) of your report with the essay, named **CW3ProPreport.pdf**. Please **DO NOT include your name, or the declaration of work** within this document, so that we can mark anonymously.

How to Submit

Submission is a two-step process: (i) upload the file, (ii) 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.
- 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 ask for help if you have a problem submitting.

5 Deadline

Please submit the report by **12:00, Fri 7th April 2023**. You are allowed to submit the declaration of work 15 minutes later.

This coursework is worth 63% of the total coursework mark: 38% for the SE tasks, and 25% for the ProP tasks. We estimate it should take each team member around 26 hours of work (20 for SE tasks, 6 for ProP task).

Borislav Ikonomov, Vidminas Vizgirda, Cristina Alexandru, 2023.