CS5030 - Software Engineering Principles

Assignment 3 – Class-wide Feedback

Assignment

Requirements specification

Assumptions, functionality and quality attributes were generally reasonable. You shouldn't make assumptions that contradict the specification. Assumptions can clarify or refine the specification.

Each requirement should be atomic- it should address a single unit of functionality or one quality. Avoid duplication in requirements. Use a consistent format for specifying them.

Requirements should be as precise as possible without making design or implementation level decisions. At this stage, you are specifying what is required and not how it should be implemented.

Functional and non-functional requirements are expressed as system requirements – "The system shall …" for mandatory requirements and "The system should …" for desirable requirements.

Non-functional requirements should focus on qualities required from the system. You should quantify them as much as possible.

It is useful to assign priorities to requirements to help with system design.

Ethical concerns

There were some good points made here.

You can consider professionalism in development, security, privacy, accessibility and preventing misuse of the system in this context. It would be good to indicate how / in which artefacts these concerns are / can be addressed.

Use case (diagram and specification)

The use case diagram was generally done well with appropriate use of relationships, including generalisation, include and extend. Extend relationships are guarded by a condition, which should be specified. The generalisation arrowhead is hollow. Consider the impact of generalisation and the use cases that become applicable to different actors.

This diagram is done from the perspective of actors and not the system. The diagram should show the system boundary and external systems (also actors) that are needed for use cases.

Meaningful and appropriate names should be used for use cases.

You don't need to deal with error conditions in this diagram.

Use case specifications were generally fine.

Each step in the main flow should be specified in active voice. Specify pre and post conditions where appropriate.

Logical software architecture

There was some variability here. You should not confuse the notation for representing the architecture with the architecture itself or a style.

Notation:

A UML diagram was required. A component diagram is appropriate for logical architecture. This is a higher level model so a fully-fledged class diagram will have too much detail. At this level, we are interested in the components in the system, how they are configured together and possible interactions among them.

Style:

It is useful to consider possible styles (both their benefits and possible disadvantages) before deriving the specific architecture. A number of styles may be able to support your functional requirements but you should also consider how different styles might help or hinder the non-functional requirements of the system. Avoid using an overly complex style unless the system domain and non-functional requirements require it. You should also check that the typical configuration of the style can be applied to the system. Some styles require additional components to manage the components required by the application.

Architecture:

You would instantiate the style to create the logical architecture for a specific system and you would represent it in the chosen notation. It is useful to show the whole system as a single component / element interacting with external systems and users but this is not sufficient. You need to decompose the system according to the chosen style at least once. You may use different styles for different levels of decomposition. You can show all the levels in one diagram if it doesn't get too cluttered. Otherwise, create separate diagrams.

You should justify your decisions.

Design (structural and interaction)

A class diagram is an appropriate notation for structural design. The overall structure and relationships were mostly OK in submissions.

It would be useful to explain how the structures in the design and the architecture correspond. The architecture is the broader structure within which the design is developed.

Remember to show navigability or multiplicity details of associations. You should use either inline attributes or associations (with multiplicity and navigability information) to indicate relationships between classes but not both.

Use inheritance where appropriate.

You should follow standard practices for good design such as encapsulation.

Sequence diagrams are good for modelling interactions. You should clearly identify the interaction being modelled.

Participants in these diagrams are objects within the system and external actors.

If objects are created during interactions, you should show create messages as well.

A better solution will showcase multiple constructs such as fragments and different types of messages.

Analysis and reflection

There was some variability here. Relevant points can include:

- How artefacts influenced one another and the traceability among them
- How the overall design satisfies the requirements of the system
- Any interesting points about the process of creating the artefacts – for eg, was this an iterative process or were you able to create each artefact in one go?
- How did the possibility of wider use in future affect your design?
- Having produced the design, if you were to do this again, would you do anything differently?

There were some good discussions of the pros and cons and the experience of using UML.

Report

Reports were generally structured well. Formatting is important when there is a long list of items or a lot of text.

It's good to get into the habit of highlighting any significant decisions and justifying them in each section. Clearly identify the notation used in each section.

The information should be presented in a logical order. The order from the spec would be fine. For eg, the logical software architecture should be described before software design.

Don't include unnecessary information. For eg, you don't need to give generic definitions of different artefacts. The information required pertains to this specific system.