CITS4401 Software Requirements and Design
Week 7 & 8: UML diagrams and Software design
Last updated on 26 April 2023

Reading: Pressman Chapter 8 (Requirements Modelling)

Questions 1:
"Software design principles are key notions that provide the basis for many different software design approaches and concepts. Software design principles include:

- abstraction;
- coupling and cohesion;
- decomposition and modularization;
- encapsulation/information hiding;
- separation of interface and implementation;
- sufficiency, completeness, and primitiveness; and
- separation of concerns." [SWEBoK Chapter 2]

In your own words give a short explanation for as many of these concepts as you can. Also, provide an example for each concept. If you are not sure about a concept then make a note and come back to it in future weeks. By the end of the unit, you should be able to explain each of the principles above.

For discussion in class. Cohesion measures the dependence among classes; Coupling measures dependencies between subsystems; Subsystems should have as maximum cohesion and minimum coupling as possible Information hiding is the idea that every module should hide aspects of its implementation - exposing only an undestandable interface.

Question 2:
When you write a program, are you doing "design"? Why or why not? If not, then what makes software design different from coding?

Probably not. If we regard design as consisting of a set of models which record enough information on our choices of subsystem decomposition, data management, and so on, for the system to be implemented according to the requirements ? then when we write code, we are not doing design. Our decisions and models might be expressed through the code ? but in most programming languages, they are expressed only implicitly. (And fairly haphazardly.) Software design normally consists of models or representations of a system, rather than code itself

Question 3:
If a software design is not a program, then what is it?

See the previous question for one answer ? we could consider design to be set of models, representing choices made, sufficient that the system can then be implemented from the models. A principle is “a comprehensive and fundamental law, doctrine, or assumption? [7]. Software design principles are key notions that provide the basis for many different software design approaches and concepts. Software design principles include abstraction; coupling and cohesion; decomposition and modularization; encapsulation/ information hiding; separation of interface and implementation; sufficiency, completeness, and primitiveness; and separation of concerns.

Question 4:
How do we assess the quality of a software design?

Many ways of assessing design quality have been suggested - did yours differ from the following? McGlaughlin ([McG91], cited in Pressman) suggests that:

- The design should implement all explicit requirements contained in the requirements model, and it must accommodate all the implicit requirements desired by stakeholders.
- The design should be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective

Bruegge and Dutoit ([Bru04], in section 7.4.7) suggest it should be:

- correct: all elements of the design should be able to be traced to some requirement
- complete: all requirements must be addressed in the design
- consistent: there are no cases where aspects of design conflict, or where design goals violate nonfunctional requirements
- realistic: it should be possible to be implemented
- readable: developers should be able to understand the design.

The lecture slides state a design should:

- be simple
- be coherent (consistent, and providing a unified picture of the system)
- adequately meet requirements
- be adaptable

SWEBOK (Chapter 2) notes that There is an interesting distinction between quality attributes discernible at runtime (for example, performance, security, availability, functionality, usability), those not discernible at runtime (for example, modifiability, portability, reusability, testability), and those related to the architecture?s intrinsic qualities (for example, conceptual integrity, correctness, completeness).

These sets of criteria aren?t identical, but capture elements of what we might consider “good design? in different ways. For instance, a system that contains extraneous

functionality would not be as simple as possible, nor would it meet Bruegge and Dutoit?s criterion of being correct (since there would be functionality not traceable to a requirement). [McG91]: McLaughlin, Robert. ?Some notes on software design: reply to a reaction.? ACM SIGSOFT Software Engineering Notes 17.2 (1992): 70. [Bru04]: Bruegge, Bernd, and Allen H. Dutoit. Object Oriented Software Engineering Using Uml, Patterns, and Java. Upper Saddle River, NJ: Pearson Education, 2004.

Question 5 :
The *E-mail Filter* system filters incoming e-mails with a whitelist (e-mails from senders on the whitelist are accepted), a blacklist (e-mails from senders on the blacklist are deleted), and the Spam-assassin tool (e-mails that do not pass this check are marked as spam). The system will run on a single-core server machine, but may be moved to a multi-core server if the load gets too high.

A brief web search reveals that there are hundreds of existing software libraries that can identify spam emails for you. But they are written in different languages and use a variety of machine learning approaches. On the other hand, you could implement your own Spam filter.

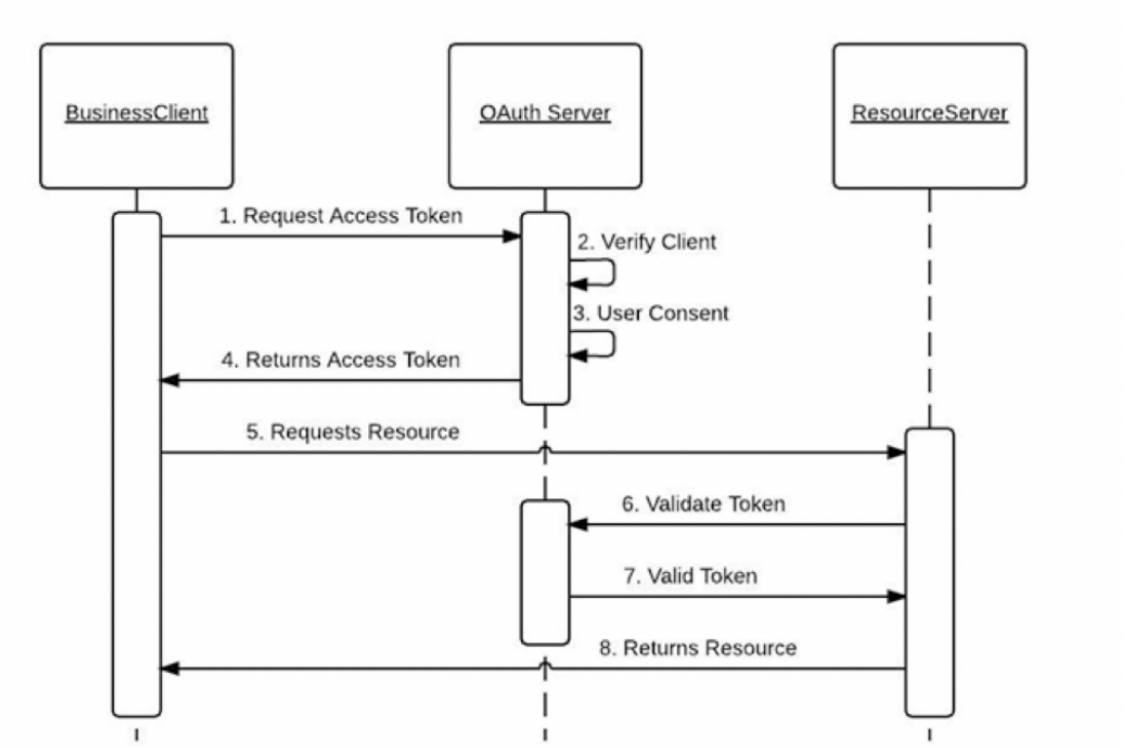
Use ***design rationale*** to document the best design decision considering the alternatives of using existing spam software or implementing your own.
Your answer should clearly identify: Issue, Proposals, Criteria, Arguments and Resolution.
Also mention any unresolved questions or assumptions you identify.

Issues

Solution: Please refer to the discussion in the workshop.

Question 6 **Reading UML sequence diagrams**

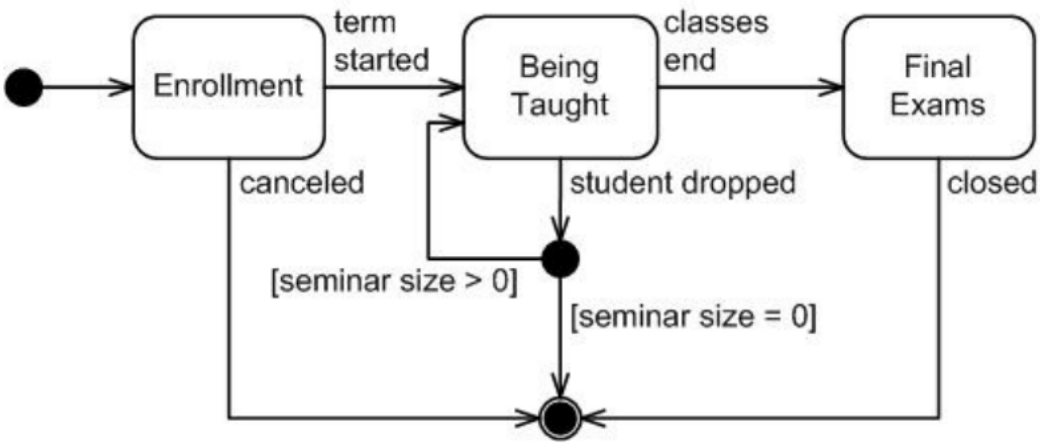
In plain English sentences, describe the scenario represented by the following UML sequence diagram. You can read more about this (real world!) scenario at https://docs.oracle.com/cd/E82085_01/160027/JOS%20Implementation%20Guide/Output/oauth.htm



Solution: There are 3 classes interacting in this diagram: a business client, an authorisation server and a resource server. First the client requests an access token, then the authorisation server verifies the client and the user consent. Next the authorisation server returns an access token to the business client. Finally the business client requests a resource from the resource server. The server responds to the authorisation server with a validatetoken message and then the authorisation server returns a valid token message. This exchange checks that the business user's token is valid. Finally the resource server returns the resource that was requested to the business client. This diagram does not include any alternate flows for invalid actions.

Question 7

In plain English sentences, describe the scenario represented by the following UML state diagram. For more details see the source for this example which includes actions for managing a waiting list and actions for state changes. https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm



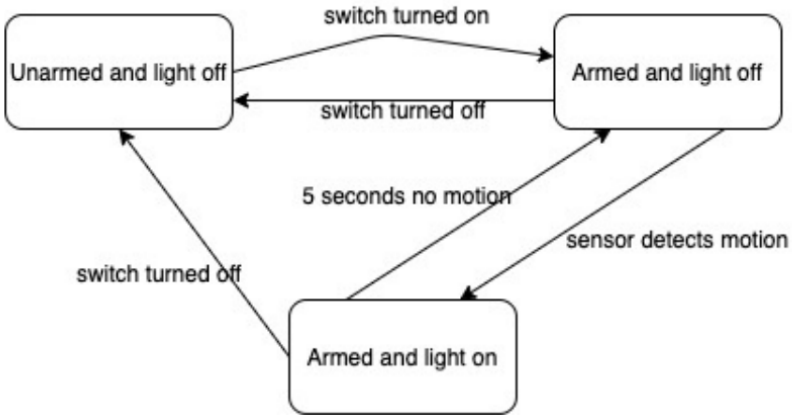
Solution

This state diagram represents the process of enrolling in a class. Initially the class is ready for enrolment. Once term has started the class changes to the being taught state. Alternatively if the class is cancelled then the state machine terminates. While being taught a student can drop the class. If the class size remains at more than 0 then the class continues to be taught, else (size=0) it terminates. When classes end then the system transitions to final exam state. On completion of the exams the class is closed.

Question 8:

Developing UML state diagrams Draw a UML state diagram to describe the states of the following security light system. A security light system has a switch and a motion sensor attached. It can be either armed or unarmed. If the switch is in the off position the light is off and the system is unarmed. When the switch is turned on, the light stays off but the system is armed. If the system is armed and the motion sensor detects movement, the light comes on. If no movement is detected for 5 seconds, the light goes off.

SOLUTION:



This diagram does not show initial state. Does that make any difference ?