Design evaluation

Sverre Hendseth

January 26, 2017

1 Specification

You are going to present your design of the lift system. The main point is to decide, and be able argue clearly, how you will solve the fault tolerance challenge ("No orders should ever be lost"). The design should be followed through at least until the system is divided into modules and the module interfaces are complete, realistic and relatively stable.

2 Evaluation Criteria

- Solution: Did you solve the problem? Are we confident that your system will handle the plthora of errors and still handle all orders?
- Modules: Are the modules you have chosen well named, and gives a reasonable partitioning of the system (do we get the feeling that dependencies/coupling between the modules are minimized and that everything is thought through?)
- Module interfaces: Do the module interfaces seem nice abstractions of the modules responsibilities?
- Presentation: Did you manage to convey your solution and your modules clearly?

The way you describe the module interface will be different depending on how modules interact/communicate. Ex. In C a module is typically described by its header-file, while in Go a list of services/incoming message types that a server should handle would typically describe its interface.

3 Presentation

We will spend 15 minutes per group (a hard maximum). Prepare a presentation aiming for 10 minutes, presenting your design:

- How you solve the fault tolerance challenge.
- Which modules you have chosen to partition the system into. (Including "interfaces").

4 Hints

Many thick books has been written about systematic approaches to software design...

Here is my take:

1. Have a brainstorm on "what the system must handle": You must handle "that a button is pressed", "that a node is killed", etc. This need not be a complete set of scenarios, but should be illustrating different aspects of the system functionality. (I would guess 4-8 scenarios should span the functionality space)
2. Divide the system into modules based on your gut feeling and discussions with your partner.
3. Map the scenarios onto the modules in a "then this must happen" manner: "The button press is detected by the X module, that notifies the event loop in the Y module. Then a message is sent ... etc." (Also here: do not give in to the temptation to make "complete" scenarios, focus on spanning the space.)
4. Draw the diagram over module interactions. Who calls who?
5. Sum up, for each module, all its incomming interactions/requests. These are the specification *of the module.*
6. Try to make a perfect module interface that fulfils the modules responibilities.

7. Move responsibilities between modules (reorganize how the system is divided into modules if necessary) so that the diagram in 4. gets fewer arrows and that the module interfaces in 6. becomes perfect abstractions.

(For a larger system this whole process can be repeated on the module level)