CITS4401 Software Requirements and Design
Semester 1, 2020
Workshop week 7 – Introduction to design

Exercises:

1. 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.

2. 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.

3. 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.

Brügge 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

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]: Brügge, Bernd, and Allen H. Dutoit. Object Oriented Software Engineering Using Uml, Patterns, and Java. Upper Saddle River, NJ: Pearson Education, 2004.