CSC444F Software Engineering I

Tutorial Assignment 4

This assignment is handed out during the tutorial of week 8 (Week of 22/10/2001)
This assignment is due at **the start of your week 10 tutorial (i.e. on 5/11/2001).**
To avoid late penalties, submit it to your TA within the first half hour of the tutorial.

Penalties

Reports submitted up to 48 hours late: -50%.
Reports submitted more than 48 hours late will not be graded.

Grading Scheme

This assignment constitutes 10% of your grade for the course.
This report is a team assignment. Each team should submit a single report, and all members of the team will receive the same grade. *See the course orientation handout for details on team grading*. You should include a short statement about which team members wrote which parts of this assignment. If some parts were joint efforts, make this clear.

Content

The assignment is to critically assess the quality of the three modules you bought at the end of phase 1, and to report on your integration effort.

You should submit the following:

1) State which modules you bought, from which teams. Give a brief (1 paragraph each) description of why you chose each of these modules from among their competitors.
2) For each of the modules you bought at the end of phase 1, a description of the **quality** of the module. Include in your report a description of:
   - the quality factors that you used to make the assessment;
   - why those factors are important to you (ie. how do they relate to *"fitness for purpose"*);
   - the metrics you used to assess these quality factors, and
   - the actual data collected for each metric.

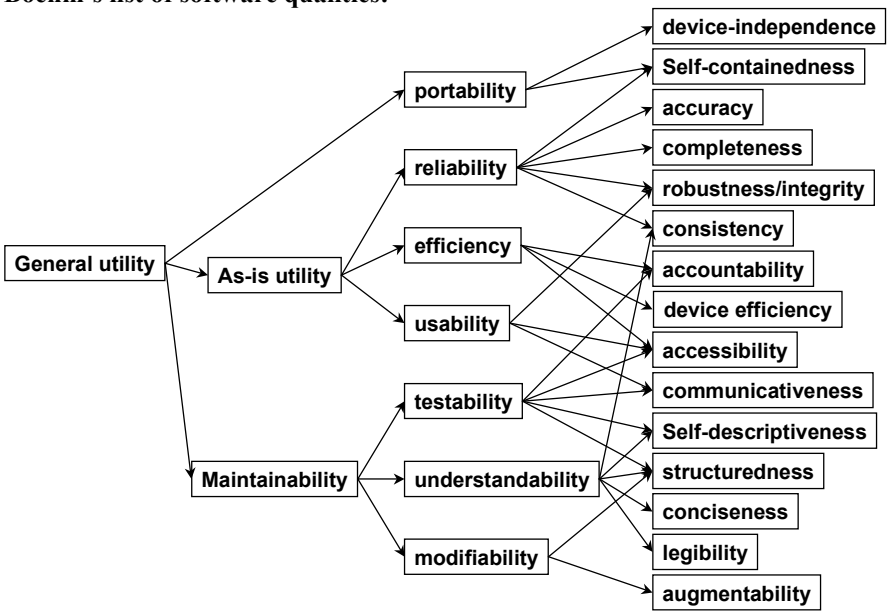
   Conclude by comparing your reasons for buying the module with your assessment of its quality – would you make the same purchasing decision again?
3) Provide a summary of the modifications you have made to the modules you bought at the end of phase 1. To do this you should define a small number of classes that characterize different types of modification you have made, based on both the original location of the problem (e.g. was it a problem in the original spec, a problem of interpreting the spec, a problem in the interface standards, a design error, a coding error, etc) and the type of the problem (e.g. missing function, extra function, wrongly implemented, misinterpretation, incorrect assumption, incorrect interface, etc). Provide summary data for numbers of modifications in each class. Then choose *one* modification from each class, and describe:
   - how you discovered the problem;
   - whether the problem could have been anticipated and avoided by improving the original specification;
   - whether the team that built the module could have predicted the need for the modification;
   - what the team that built the module could have done (if anything) to make the modification easier.
4) Briefly describe your integration testing strategy, and describe (in detail) four of the test cases you used for integration testing, indicating which tests passed and which failed. What other kinds of testing did you use? What are the strengths and weaknesses of your integration testing strategy?

Background Information

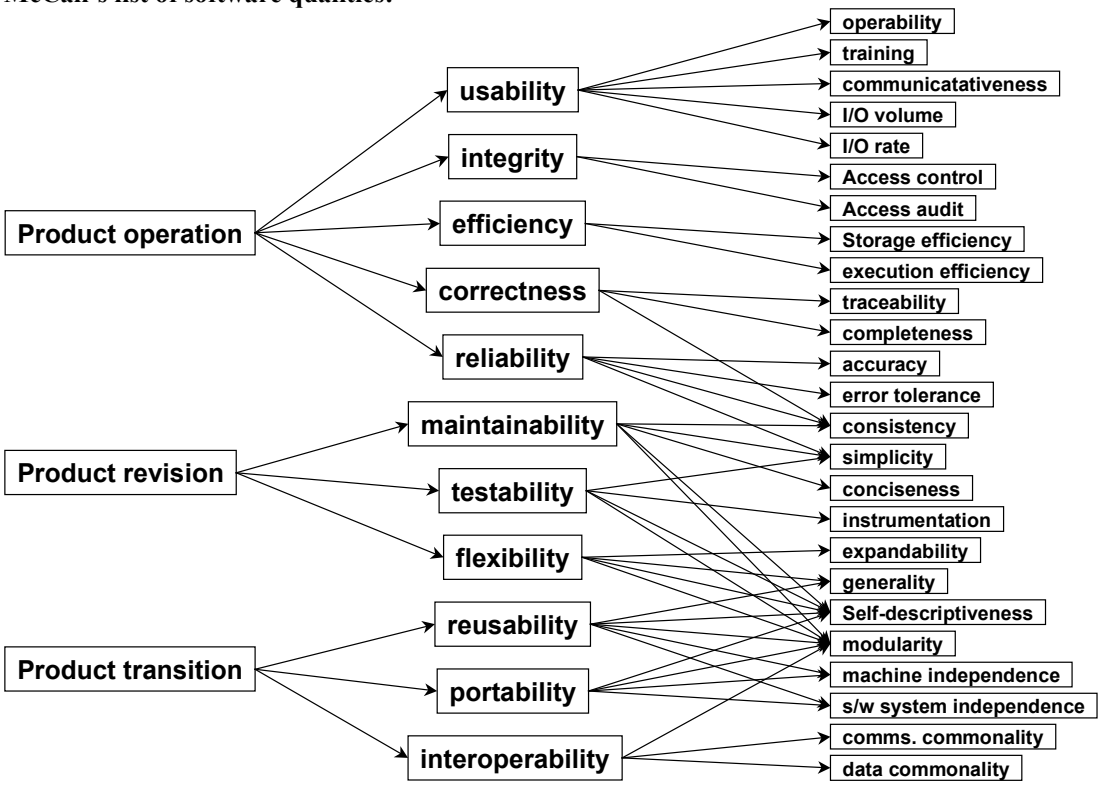
The ultimate measure of design quality is fitness to purpose. This means that to measure software quality we have to understand what the purposes are for which it is intended. Note this also means that quality is not a measure of software in isolation; it is a measure of the relationship between software and its application domain. This means that an assessment of its quality depends on the context, and if you change the context, the assessment of quality may change – software that's good for one purpose may not be so good for another purpose.

Measurement of quality generally starts by identifying a set of critical quality factors, and then refining these down to measurable attributes of the software. There is no agreed way of doing this. Different quality assessors disagree about what various measurable attributes of software actually tell you about its quality. Two well-known quality factor lists are those of Boehm and McCall. They differ, both on the factors they identify and on the mappings between higher level and lower level factors. Note that both stop before they get to measurable attributes – each of their bottom level factors still needs to be mapped onto *measurable* aspects of the software (i.e. things that you can actually count!).

Boehm's list of software qualities:



McCall's list of software qualities:



See also: Lecture notes for lectures 12 and 13