2010.2 OOP Handout 1 : Factors of Software Quality

* What is a good software (program)?

- correct : correctly satisfy user requirement/specification
- efficient (space / time) , HW utilization
- easy to maintain (software evolution)
  - easy to read (readablity)
  - easy to debug , easy to test
  - easy to modify
  - easy to extend(extensibility) , scalability
  - easy to reuse (reusability)
  - easy to port (portability) , platform-independent
- stable (stability), reliable (reliability)
- easy to use (usability) , easy to run
- concise (conciseness)
- secure (security)
- complete (completeness)
- consistent (consistency)
- well documented
- well structured

* Factors of Software Quality

Understandability
Are variable names descriptive of the physical or functional property represented? Do uniquely recognisable functions contain adequate comments so that their purpose is clear? Are deviations from forward logical flow adequately commented? Are all elements of an array functionally related?...

Completeness
Are all necessary components available? Does any process fail for lack of resources or programming? Are all potential pathways through the code accounted for, including proper error handling?

Conciseness
Is all code reachable? Is any code redundant? How many statements within loops could be placed outside the loop, thus reducing computation time? Are branch decisions too complex?

Portability
Does the program depend upon system or library routines unique to a particular installation? Have machine-dependent statements been flagged and commented? Has dependency on internal bit representation of alphanumeric or special characters been avoided? How much effort would be required to transfer the program from one hardware/software system or environment to another?

Consistency
Is one variable name used to represent different logical or physical entities in the program? Does the program contain only one representation for any given physical or mathematical constant? Are functionally similar arithmetic expressions similarly constructed? Is a consistent scheme used for indentation, nomenclature, the color palette, fonts and other visual elements?

Maintainability

Has some memory capacity been reserved for future expansion? Is the design cohesive—i.e., does each module have distinct, recognizable functionality? Does the software allow for a change in data structures (object-oriented designs are more likely to allow for this)? If the code is procedure-based (rather than object-oriented), is a change likely to require restructuring the main program, or just a module?

Testability

Are complex structures employed in the code? Does the detailed design contain clear pseudo-code? Is the pseudo-code at a higher level of abstraction than the code? If tasking is used in concurrent designs, are schemes available for providing adequate test cases?

Usability

Is a GUI used? Is there adequate on-line help? Is a user manual provided? Are meaningful error messages provided?

Reliability

Are loop indexes range-tested? Is input data checked for range errors? Is divide-by-zero avoided? Is exception handling provided? It is the probability that the software performs its intended functions correctly in a specified period of time under stated operation conditions. but there could also be a problem with the requirement document...

Efficiency

Have functions been optimized for speed? Have repeatedly used blocks of code been formed into subroutines? Has the program been checked for memory leaks or overflow errors?

Security

Does the software protect itself and its data against unauthorized access and use? Does it allow its operator to enforce security policies? Are security mechanisms appropriate, adequate and correctly implemented? Can the software withstand attacks that can be anticipated in its intended environment?

User's perspective

In addition to the technical qualities of software, the end user's experience also determines the quality of software. This aspect of software quality is called usability. It is hard to quantify the usability of a given software product. Some important questions to be asked are:
- Is the user interface intuitive (self-explanatory/self-documenting)?
- Is it easy to perform simple operations?
- Is it feasible to perform complex operations?
- Does the software give sensible error messages?
- Do widgets behave as expected?
- Is the software well documented?
- Is the user interface responsive or too slow?

Also, the availability of (free or paid) support may factor into the usability of the software.