Review of Topics

Software Engineering (SE)

Recall that software engineering is a multi-pronged attack on the problems of producing reliable software that meets clients needs.

Some of the techniques are for the individual, some are for groups, and some are for entire companies.

Some of the techniques are technical, some are economic, some are managerial, some are social, and some are psychological.

Topics

- **Myths of Software Development**
- **Information Hiding**
- **Object Orientation**
- **Testing**
- **Documentation**
- **Requirements Engineering**
- **Software Project Management**

Purpose of Review

The purpose of this review is to remind you of the salient points of each topic, now that you have had a chance to digest them and later topics.

Common Themes

The common theme running through many, but not all, topics is that of *avoiding harmful effects of inevitable changes*

This can be done by two different approaches:

- **making changes less likely, and/or**
- **limiting the effect of each change.**

This theme is cost driven, as it is known that a majority of the cost of software these days is in its maintenance.

Myths of Software Development

To begin the process of unlearning bad habits

- **A lot of the folklore does not match reality.**
- **Unfortunately, decisions have been made based on folklore rather than reality.**
- **Only by knowing reality can good decisions be made.**
- **New reality is learned all the time, and it is up to *you* to be questioning assumptions to learn the realities.**

Abstract Data Typing -1

With an emphasis on being able to modify implementations

- **Abstraction is hiding implementation details so that they can be changed without affecting software that *uses* the implemented functions or data.**
- **Abstract data typing is hiding the implementation of data structures while providing a full set of functions and procedures with which to manipulate the data.**

Abstract Data Typing -2

- **Some languages, e.g., Ada, provide a linguistic feature that enforces hiding by means of encapsulation.**
- **The important thing is the concept, so that even if one does not have a hiding-enforcing language, one can build and use abstract data types.**

Information Hiding -1

Both a method for decomposing software and a demonstration that performance is independent of modularization

- **Information hiding is a method for decomposing software into modules such that at most one module, that implementing a single concept, has to be changed when the implementation of the concept is changed, and in particular the rest of the program, that *uses* the concept does not have to be changed.**

Information Hiding -2

- **Parnas suggests identifying good decompositions by considering as many implementation changes as possible and seeing how many modules are affected by each change.**
- **Parnas suggests the use of abstract data types to make change-withstanding modules for data.**

Information Hiding -3

- **Parnas demonstrates that the running object program generated from a modular program can be *identical* to that generated from a program designed specifically around one implementation to be as efficient as possible for a given situation; hence use of modularity can carry *no* run-time performance penalty.**

Object Orientation -1

With an emphasis on inheritance

- **Object orientation is structuring software into abstraction-modules such that each instantiation of an abstraction corresponds to an object in the real-world domain modeled by the software.**
- **Modules structured this way resist change simply because the real-life situation changes very slowly compared to software.**

Object Orientation -2

- **One key tool in building real-life-modeling objects is inheritance that allows a subclass to inherit properties, i.e., function and data, from another more general class.**
- **Inheritance aids change resistance by allowing grouping of properties that occur in only some members of a class into a subclass.**

Testing -1

Particularly of modular programs

- **Testing is running a program with contrived data for the purpose of *finding* errors, and is not attempting to demonstrate that the program is correct.**
- **Individual modules and whole programs are tested against data generated from both the specifications and the internal structure.**

Testing -2

- **Integration testing should proceed module-by-module in order to be able to easily isolate the modules involved in discovered errors.**

Documentation

Of, by, and for modular programs

- **It is impossible to carry out totally rational design methods.**
- **It is possible to write documentation as if the design were totally rational, i.e., to fake it on the documentation.**
- **Such documentation, aimed for the maintainers should show the modules, their semantics, their interfaces, and the implementation secrets that they hide.**

Requirements Engineering -1

Using modular requirements to an advantage

- **The big problem is dealing with the requirements engineer (RE)'s ignorance of client's domain and the client's ignorance of computing.**
- **Information hiding played at the domain level can be used to hide the RE's ignorance so that he or she can work intelligently with its abstractions.**

Requirements Engineering -2

- **Ignorance of the client's domain helps RE to avoid falling into the tacit assumption tarpit.**

Software Project Management -1

The way it ought to be and how to get it going

- **In any project inolving more than one person, management is needed to control the nontechnical, human interaction, issues that can dominate the technical aspects if let be.**

Software Project Management -2

- **While there is a large collection of seemingly unrelated management techniques, the key thing is to *understand* the nontechnical issues and how explosive they can be.**
- **A technique is used, not because it *should* work, but because it *does* work.**

C++ is like teenage sex

- **It is on everyone's mind all the time.**
- **Everyone talks about it all the time.**
- **Everyone thinks everyone else is doing it.**
- **Almost no one is really doing it.**
- **The few that are doing it are:**
  - **doing it poorly,**
  - **sure it will be better next time, and**
  - **not practicing it safely.**

— Graffiti found in a toilet stall in the Faculty of Computer Science, Technion, November, 1993

C++ and teenage sex

It should say, “Programming in C++ is like teenage sex”.

And even *that’s* wrong!

It should really say, “Object-oriented programming in C++ is like teenage sex”.

It is really inappropriate to equate OOP and C++; you can do OOP without C++, *and* you don’t need to do OOP when you use C++.