Some Software Engineering Principles

D. L. Parnas
What is Software Engineering

Software Engineering = Multi-Person construction of multi-version programs

- Decomposing program projects into manageable work assignments
- Precise specification of the work to be done
- Design for extension
- Design abstract interfaces for modules
Challenges of Multi-Person Programming

- How to divide the work across multiple programmers and allow them to work efficiently
- Specifying the behavior and interfaces associated with the software components
- Communication: To the user and about the inter-component (modules) dependencies
Design and Programming Challenges in Multi-Version Programming Projects

- Design so that the programs can be easily modified

- Design of programs with useful “subsets”
  - Allows the “parts” to be extended and/or replaced without touching the remainder of the program

- Design for extension
The Structure of Programs

- The structure of the system shows the system divided into parts and specifies the connections between the parts
  - Sound like a software architecture definition?

- A particular system can be divided many different ways into “parts”
  - Sound like the different views of the software architecture?
Connecting the Parts

- Must be careful to specify the connections between the parts
- Parnas Definition: The connections between the program parts are the assumptions that the parts make about each other
  - What are the system properties that the part is required to guarantee?
  - What properties does the part expect from the other system parts?
- Allows for the answering of important questions
  - What parts can be changed without affecting the other parts?

Recommended Practice: Make the connections between the parts contain as little information as possible.
What is a Module?

- Often a “Module” is used as a buzzword
- A Module is a part that can be combined with other parts to form the complete system
- Determining the module structure is dependant on the criteria used to decompose the system into modules
- In this paper, a module is a work assignment
  - A unit of change – each change should be limited to as few modules as possible
  - A piece of work that can be implemented by an individual or a small group of programmers

This is a Software Engineering perspective on a Module
Creating Well-Structured Systems

- **Decomposition**: Division of the project into Modules (work assignments)
- **Specification**: Specifying the behavior and interfaces of the Modules
Specifying the Modules

◆ The “Flowchart” approach
  ■ Useful for solo projects, but bad for multi-person projects
  ■ Often results in modules that have significant dependencies on other modules

◆ Information Hiding Approach
  ■ Specify modules by considering assumptions that are likely to change
  ■ Design a module to hide the secrets that are based on the assumptions
  ■ Leads to modules that have abstract interfaces that are unlikely to change
Module Specification

- Precisely define the assumptions that the designers of a module can make about the other modules
  - Must be careful to do the specification at the correct level – too much and too little information is bad

- Parnas Suggestion: The specification should capture the externally visible behavior and interfaces provided by a module
  - Sounds a little like a use-case
Handling Errors

An information hiding approach makes dealing with errors more difficult

- Errors cannot be expressed in terms of internal structures for which the other modules have no knowledge
- Information about the error is embedded in other modules and may be difficult to abstract

Parnas Suggestion: Use the unexpected event technique for error handling described in his other papers
Hierarchically Structured Systems

- Hierarchies are often used as Buzzwords
- The “uses” hierarchy is useful in SE
  - Uses Relation: \( P_1 \text{ uses } P_2 \) if a working copy of \( P_2 \) is required for \( P_1 \) to meet its specification
  - If the uses relation is a hierarchy it will be easy to identify useful program subsets
  - If the uses relation is not a hierarchy (it contains loops) then the program may require all of its parts before the program works
- Parnas Suggestion: Design the uses relation early – don’t leave it to the individual programmers during construction
Designing Abstract Interfaces

Design the interfaces in 2 phases:
  - List the assumptions believed unlikely to change with respect to the modules responsibility
  - Specify the interface so that their implementation covers the set of assumptions
Summary

- Upfront work is important to help improve the outcome of a software project
  - This is the engineering part of SE
- This paper provides a good summary of the other concepts that we have investigated this term