Outline

- Software Life Cycle
- Modularity
- Design Methodologies
- Tools of the Trade
- Testing
- Documentation

Software Engineering

- process of software development and maintenance
- large projects
- resource management and planning
- development methodologies

Lifecycle

- software as a product
- not physical so doesn’t wear out
- errors and changes lead to modifications

Modification

- bulk of lifecycle is modification – use
- ease of modification is important
- proper development methods contribute to ease of modification (example: use of constants)
- invest time and effort in good design
Development Phase

- Analysis
  - requirements, specification
- Design
  - design documents
- Implementation
  - programs
- Testing

Analysis

- Needs assessment: automate or not?
- Based on application domain
- Analyze current method: inputs, outputs, changes
- Understand what and how of current method
- Requirements: What the system shall do
- Specifications: How

Design

- Develop technical details based upon specifications
- Develop and organize modules, plan the structure of the system
- Necessary for large systems and team development
- Personnel and resource allocation

Implementation

- actual development of each module (writing source code)
- development of databases
- user interface components

Testing

- does system conform to specifications?
- module-level tests
- integration test
- development of test data
- testing methodologies still being researched

Waterfall Model

- strict sequential application of the phases
- changes cost less the earlier they are done in the process
- appropriate for well-understood systems
- need more flexible methodology for novel systems
New Methods

- CASE tools (computer aided software engineering)
- make modifications easier and faster
- can generate code from specifications
- Prototyping and Iterative Design
- Evolutionary vs. Throwaway prototypes
- Extreme programming

Outline

- Software Life Cycle
  - Modularity
- Design Methodologies
- Tools of the Trade
- Testing
- Documentation

Modular Design

- modular design = abstraction + organization
- structure charts show organization of modules, data flow
- modules represented by rectangles
- hierarchy represented by arcs between rectangles
- data flow by labeled arcs

Example Structure Chart

Example Class Diagram

Advantages

- independent modules = concurrent development
- ease of module tests
- stub: routine that does nothing but allow its caller to be tested
- driver: routine that only calls a module for testing
**Coupling**

- Goal is a system that is easy to modify
- Independent modules are easier to modify
- For modules to form a system they MUST be connected
- Coupling = degree of connectedness of modules
- Low coupling → high independence → modifiability

**Control Coupling**

- Module passes control to another on
- Link = call-return relationship

```
PROCEDURE A()
 statement1;
 statement2;
 call B();
 ...
 END
```

**Data Coupling**

- Number of parameters exchanged
- Symptom of poor design of modules
- A module should not handle data that it doesn’t need

**Structure Chart showing Data Coupling**

**Collaboration Diagram**

**Global Variables**

- Modules local variables are encapsulated, or protected from access by other modules
- Global variable is a variable that all modules can access
- Thus any module could change the value of a global variable
- THE PROBLEM is that such changes are HIDDEN!
 Explicit = Modifiable

• when you need to make changes, it's easier if you can see what things to change
• implicit coupling (global variables) make it hard to see what modules are related

CheckPrivileges(Password, Privileges) vs. CheckPrivileges(Password)

Side-effects via Parameters

• not limited to global variables
• if a procedure changes a parameter unexpectedly, that is a side effect.

Display(Privilege) – changes privilege level inside,

If now we call Modify(Privilege), we might allow modifications to be made when they shouldn't

Cohesion

• internal relatedness of a module
• similar to cohesion in a paragraph

all sentences in a paragraph must support the topic sentence
no sentence in a paragraph talks about something other than the topic

Types of Cohesion

• logical: all activities in the module are logically similar
• functional: one task per module
• no clear step-by-step rules for how to determine cohesion
• depends on point of view or concern

Outline

- Software Life Cycle
- Modularity
- Design Methodologies
- Tools of the Trade
- Testing
- Documentation
Development Tools and Methodologies

- top-down design
- bottom-up design
- dataflow diagrams (DFDs)
- entity-relationship diagrams (ER diagrams)
- data dictionaries

Top-down Design

- similar to stepwise refinement
- first restate the problem: WHAT
- list the major tasks involved: HOW
- supervisor module controls interaction among subordinate modules, each of which performs a single task

Bottom-up Design

- identify individual tasks; solve those
- assemble many small solutions into a system to solve a larger problem
- not all applications have a hierarchical nature (example: simulation of networks)
- building block approach: small solutions can be used in many different systems

How do we come up with ...

- how do you know what the supervisor module is? ask: what does the system do?
- how do you know what the subordinate modules are? ask: what are the general steps?
- not an immediate answer, but a PROCESS of discovering and refining an answer
- use of DFDs, ER diagrams can help

Outline

- Software Life Cycle
- Modularity
- Design Methodologies
- Tools of the Trade
- Testing
- Documentation

Dataflow Diagrams (DFDs)

- show how data moves through a system and how it is created, transformed or consumed
Entity-relationship Diagram

- entities: objects manipulated by the system
- relationship: how an object is related to another

Example: system = university
relationships:
- a teacher teaches many classes and a class is taught by only one teacher
- a student can take many classes and a class can be taken by many students

ER Diagrams

Example

![ER Diagram](Image)

Use of DFD and ER

- DFDs allow us to analyze how data are transformed by the components of a system
- ER allow us to study the structure of the data (useful also in database design)
- both help to clarify what the modules of a system should be

Data Dictionaries

- single authoritative source of information about objects and data in the system
- examples: names of variables, types, modules used in
- explicit representation of all data items can help analyst establish precisely what a user wants
- can reveal inconsistencies in nomenclature or use of data among different teams in the project (example: team 1 uses PartID, team 2 uses PartNumber)

Outline

- Software Life Cycle
- Modularity
- Design Methodologies
- Tools of the Trade
- Testing
- Documentation

Testing Methodologies

- Pareto principle
- basis path testing
- glass-box testing
- black-box testing
- boundary value analysis
- beta testing
Outline

- Software Life Cycle
- Modularity
- Design Methodologies
- Tools of the Trade
- Testing

Documentation

- 2 types: user & system
- user documentation: learn how to use software
- tutorial: features available, how to use them
- installation: how to install software on the system
- reference: details of available features; usually in alphabetical order

System Documentation

- describes software so it can be maintained
- types of documentation: system specifications, source code + comments, test suites, DFDs, ERs, structure charts, notes, memos
- problem: need to make changes; done only in source code, not the other documentation
- solutions: CASE tools, prototyping