Software Engineering
CSE470
(Spring 2002)

Instructor:
Dr. W. McUmber
mcumber@cse.msu.edu
3100 EB

Acknowledgments

● G. Coombs
● L. Dillon
● M. Heimdahl
● R. Stephenson
● National Science Foundation:
  ■ VESL (Visions of Embedded Systems Laboratory)
  ■ CDA-9700732
● Tony Torre, Detroit Diesel Corp.

What is Software Engineering???

● The study of systematic and effective processes and technologies for supporting software development and maintenance activities
  ■ Improve quality
  ■ Reduce costs

Why is software engineering needed?

● To predict time, effort, and cost
● To improve software quality
● To improve maintainability
● To meet increasing demands
● To lower software costs
● To successfully build large, complex software systems
● To facilitate group effort in developing software

Who Needs Software Engineering?

Show me a business in the U.S. that doesn’t use Software

So....

Everyone!

Historical Perspective

● 1940s: computers invented
● 1950s: assembly language, Fortran
● 1960s: COBOL, ALGOL, PL/1, operating systems
  1969: First conference on Software Eng
● 1970s: multi-user systems, databases, structured programming
**Historical Perspective (cont.)**

- 1980s: networking, personal computing, embedded systems, parallel architectures
- 1990s: information superhighway, distributed systems, OO in widespread use.
- 2000s: virtual reality, voice recognition, video conferencing, global computing, ...

**Why is software so expensive?**

- Hardware has made great advances
- But, software has made great advances ...
- We do the least understood tasks in software
- When task is simple & understood, encode it in hardware
- Demand more and more of software

**Size of programs continues to grow**

- **Large**: 5 years, 100s of programmers, 1M LOC
  - MS Word, Excel
- **Very large**: 10 years, 1000s of programmers, 10M LOC
  - Air traffic control,
  - Telecommunications, space shuttle
- **Unbelievable**: ?, years, ? programmers
  - W2K 35M LOC
  - Missile Defense System 100M LOC?

**Hardware Costs vs Software Costs** (% of overall costs)

- h/w costs vs s/w costs

**Size of programs continues to grow**

- **Trivial**: 1 month, 1 programmer, 500 LOC,
  - Intro programming assignments
- **Very small**: 4 months, 1 programmer, 2000 LOC
  - Course project
- **Small**: 2 years, 3 programmers, 50K LOC
  - Nuclear power plant, pace maker
- **Medium**: 3 years, 10s of programmers, 100K LOC
  - Optimizing compiler

**What’s the problem?**

- Software cannot be built fast enough to keep up with
  - H/W advances
  - Rising expectations
  - Feature explosion
- Increasing need for high reliability software
What’s the problem?

- Software is difficult to maintain “aging software”
- Difficult to estimate software costs and schedules
- Too many projects fail
  - Ariane Missile
  - Denver Airport Baggage System
  - Therac

Goals of this Course

- Expose you to some of the problems typically encountered in software eng
- Expose you to some of the techniques that have been found to be effective
  - Requiring more rigor
  - Often appearing “obvious” (but only after being learned)

Overview of Course

- Emphasis on analysis and design
- Learn/apply new techniques for software development
- Learn to work with a group
- Improve technical writing skills
- Become up to date on current trends in SE
- Explore presentation media and techniques

Structure of Course

- (Short) assignments over readings
- In lab assignments (various SE tools)
- Homework
- Quizzes
- Group projects (prototype, analysis, design)
- One hour exam
- Presentations: oral presentations, prototype demos

Software Engineering Phases

- Definition: What?
- Development: How?
- Maintenance: Managing change
- Umbrella Activities: Throughout lifecycle
The “Standard” Development Cycle

- Requirements definition and analysis
  - Developer must understand
    - Application domain
    - Required functionality
    - Required performance
    - User interface

- Design
  - Allocate resources
  - Estimate costs
  - Define work tasks
  - Define schedule

- Coding
  - Develop code for each module
  - Unit testing

- Testing

Definition

- System analysis
  - Allocate system resources to
    - Hardware
    - Software
    - Users

Development

- Software design
  - User interface design
  - High-level design
    - Define modular components
    - Define major data structures
  - Detailed design
    - Define algorithms and procedural detail

Development (cont.)

- Coding
  - Develop code for each module
  - Unit testing

- Integration
  - Combine modules
  - System testing

Maintenance

- Correction - Fix software defects
- Adaptation - Accommodate changes
  - New hardware
  - New company policies
- Enhancement - Add functionality
- Prevention - make more maintainable
**Umbrella Activities**

- Reviews - assure quality
- Documentation - improve maintainability
- Version control - track changes
- Configuration management - integrity of collection of components

**Software Engineering Costs**

- Maintenance
- Development
- Definition

**Relative Costs to Fix Errors**

This is why software process pays off

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Design</th>
<th>Coding</th>
<th>Testing</th>
<th>Delivery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is why software process pays off

**What’s a “Methodology” or “Process Model”**

- Set of activities, notations, tools, in defined sequence.
- Goal: Order, predictability, quality, cost control
- Follows requirements=>design=>coding, etc. sequence (usually)
- Usually defines phases or steps
- Often has notations
- Sometimes has tools

**Waterfall Process Model**

**Prototyping Process Model**
When to use prototyping?

- Help the customer pin down the requirements
- Concrete model to "test out"
- Often done via the user interface
- Explore alternative solutions to a troublesome component
  - e.g., determine if an approach gives acceptable performance
- Improve morale
  - Partially running system provides visibility into a project

*NEVER Press a prototype into production*

---

**Spiral Process Model**

![Diagram of the Spiral Process Model]

Planning  →  Risk Analysis  →  Customer Evaluation  →  Engineering

---

**Process Models**

- Idealized views of the process
- Different models are often used for different subprocesses
  - may use spiral model for overall development
    - prototyping for a particularly complex component
    - waterfall model for other components

See Evolution of the Frameworks Quagmire, Computer, July 2001, p.34

---

**Capability Maturity Model**

- **Level 1: Initial**
  - success depends on people
- **Level 2: Repeatable**
  - track cost, schedule, functionality
- **Level 3: Defined**
  - use standardized processes
- **Level 4: Managed**
  - collect detailed metrics
- **Level 5: Optimizing**
  - continuous process improvement
  - "built-in" process improvement

---

**Why is software development so difficult?**

- Communication
  - Between customer and developer
    - Poor problem definition is largest cause of failed software projects
  - Within development team
    - More people = more communication
    - New programmers need training
- Project characteristics
  - Novelty
  - Changing requirements
    - 5x cost during development
    - up to 100x cost during maintenance
  - Hardware/software configuration
  - Security requirements
  - Real time requirements
  - Reliability requirements

---

**Why is software development difficult? (cont.)**

- Personnel characteristics
  - Ability
  - Prior experience
  - Communication skills
  - Team cooperation
  - Training
- Facilities and resources
  - Identification
  - Acquisition
- Management issues
  - Realistic goals
  - Cost estimation
  - Scheduling
  - Resource allocation
  - Quality assurance
  - Version control
  - Contracts
Summary

- Software lifecycle consist of
  - Definition (what)
  - Development (how)
  - Maintenance (change)
- Different process models concentrate on different aspects
  - Waterfall model: maintainability
  - Prototype model: clarifying requirements
  - Spiral model: identifying risk
- Maintenance costs much more than development

Bottom Line

- U.S. software is a major part of our societal infrastructure
  - Costs upwards of $200 billion/year
- Need to
  - Improve software quality
  - Reduce software costs/risks

Systems Engineering

- Computer System Engineering is a problem-solving activity.
- Itemize desired system functions
- Analyze them
- Allocate functions to individual system elements
- Systems Analyst (computer systems engineer)
  - Start with customer-defined goals and constraints
  - Derive a representation of function, performance, interfaces, design constraints, and information structure
  - That can be allocated to each of the generic system elements (i.e., Software, Hardware, People, database, documentation, procedures)
- Focus on WHAT, NOT how.

Criteria for System Configuration: Technical

Criteria for allocation of function and performance to generic system elements:
- Technical Analysis: existence of necessary technology, function and performance assured, maintainability
- Environmental Interfaces: (proposed configuration integrate with external environment, interoperability)
- Off-the-shelf options must be considered.
- Manufacturing evaluation: (facilities and equipment available, quality assured?)
Criteria for System Configuration: Business Issues

Criteria for allocation of function and performance to generic system elements:
- Project Considerations: (cost, schedules, and risks)
- Business Considerations: (marketability, profitability)
- Legal Considerations: (liability, proprietary issues, infringement?)
- Human issues: (personnel trained? political problems, customer understanding of problem)

Hardware and Hardware Engineering

- Characteristics:
  - Components are packaged as individual building blocks
  - Standardized interfaces among components
  - Large number of off-the-shelf components
  - Performance, cost, and availability easily determined/measured
  - Hardware configuration built from a hierarchy of “building blocks.”

Software and Software Engineering

- Function may be the implementation of a sequential procedure for data manipulation
- Performance may not be explicitly defined (exception in real-time systems)
- Software element of computer-based system consists of two classes of programs, data, and documentation
  - Application Software:
    - Implements the procedure that is required to accommodate information processing functions
  - System Software:
    - Implements control functions that enable application software to interface with other system elements

Three high-level phases of Software Engineering

- Definition phase:
  - Software planning step
  - Software Project Plan
    - Scope of project
    - Cost and schedule estimates
  - Requirements Specification
    - System element allocated to software is defined in detail.
    - Formal information domain analysis to establish models of information flow and structure required to produce specification
    - Prototype of software is built and evaluated by customer
    - Performance requirements or resource limits defined in terms of software characteristics
  - Definition and Requirements must be performed in cooperation
Third Phase of Software Engineering

- Development Phase:
  - Translate set of requirements into an operational system element
  - Design: Design Specification
  - Coding (appropriate programming language or CASE tool)
  - Should be able to directly trace detail design descriptions from code.
- Verification, release, and maintenance phase:
  - Testing software: Testing Plan
  - to list maximum number of errors before shipping
  - Prepare software for release: Quality Assurance
  - Maintain software throughout its lifetime

Structured Design: Design Issues

- Modularity Criteria:
  - Decomposability: decompose large problem into easier to solve subproblems
  - Composability: how well modules can be reused to create other systems
  - Understandability: how easily understood without other reference info
  - Continuity: make small changes and have them reflected in corresponding changes in one or a few modules
  - Protection: architectural characteristic to reduce propagation of side effects of a given error in a module.

Design Issues

- Basic Design Principle:
  - Linguistic modular units: correspond to syntactic units in implementation language
  - Few interfaces: minimize the number of interfaces between modules
  - Small interfaces (weak coupling): minimize amount of info moving across interfaces
  - Explicit interfaces: when modules do interact, should be in obvious way
  - Information hiding: all info about module is hidden from outside access

“Uses” Relation

Design Heuristics

- Evaluate “First-cut” program structure
  - Reduce coupling:
  - Improve cohesion
  - Use exploding: common process exists in 2 or more modules
  - Use imploding: if high coupling exists, implode to reduce control transfer, reference to global data, and interface complexity
  - Minimize Structures with high fan-out: Strive for Fan-in as depth increases
  - Keep scope effect of a module within scope of control of that module effect of module should be in deeper nesting
  - Evaluate module interfaces:
    - Reduce complexity
    - Reduce redundancy
    - Improve consistency

OOD is going to help us with these issues
Design Postprocessing

- After Transaction or transform analysis: complete documentation to be included as part of architectural design
- Processing narrative for each module
- Interface description for each module
- Definition of local and global data structures
- Description of all design restrictions
- Perform review of preliminary design
- "optimization" (as required or necessary)

Design Optimization

- **Objectives:**
  - Smallest number of modules (within effective modularity criteria)
  - Least complex data structure for given purpose
  - Refinement of program structure during early design stages is best
  - Time-critical applications may require further refinements for optimizations in later stages (detailed design and coding)