

## Software Engineering


### CSE470

(Spring 2002)

---

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

01-intro 1




## 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.

01-intro 2




## What is Software Engineering ???

---

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

01-intro 3




## 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

01-intro 4



## Who Needs Software Engineering?


---

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

↓ So.....

**Everyone!**

01-intro 5



## 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

01-intro 6



## Historical Perspective (cont.)

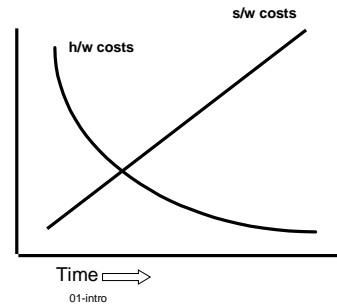
- 1980s: networking, personal computing, embedded systems, parallel architectures
- 1990s: information superhighway, distributed systems, OO in widespread use.
- 2000s: virtual reality, voice recognition, video conferencing, global computing, ...

01-intro

7



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



01-intro

8



## 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

01-intro

9



## 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

01-intro

10



## 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?

01-intro

11



## 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

01-intro

12



## 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

01-intro

13



## 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)

01-intro

14



## 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

01-intro

15



## 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

01-intro

16



## Software Engineering

A Brief Introduction

01-intro

17

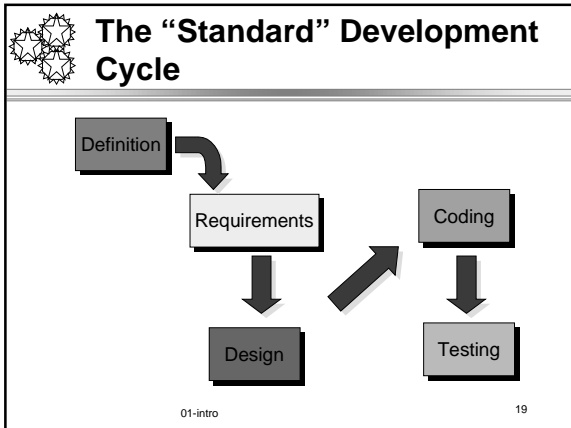


## Software Engineering Phases

- Definition: What?
- Development: How?
- Maintenance: Managing change
- Umbrella Activities: Throughout lifecycle

01-intro

18



## Definition

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

01-intro 20

## Definition (cont.)

\*

- Project planning
  - Allocate resources
  - Estimate costs
  - Define work tasks
  - Define schedule
- System analysis
  - Allocate system resources to
    - ◆ Hardware
    - ◆ Software
    - ◆ Users

01-intro 21

## Development

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

01-intro 22

## Development (cont.)

- Coding
  - Develop code for each module
  - Unit testing
- Integration
  - Combine modules
  - System testing

01-intro 23

## Maintenance

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

01-intro 24



## Umbrella Activities

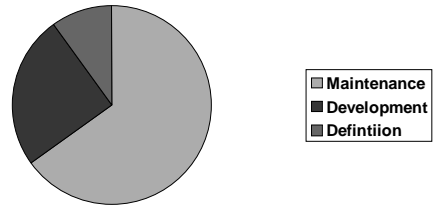
- Reviews - assure quality
- Documentation - improve maintainability
- Version control - track changes
- Configuration management - integrity of collection of components

01-intro

25



## Software Engineering Costs

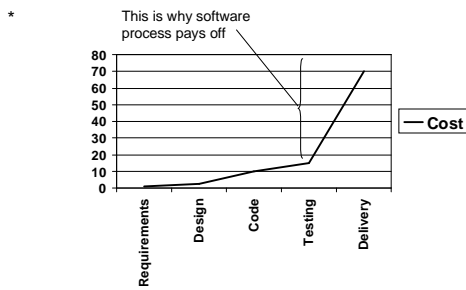


01-intro

26



## Relative Costs to Fix Errors



01-intro

27



## What's a "Methodology" or "Process Model"

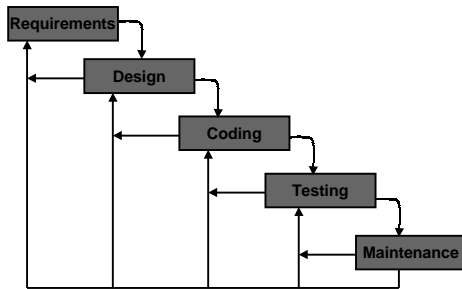
- Set of activities, notations, tools, in defined sequence.
- Goal: Order, predictability, quality, cost control
- Follows requirements=>design=>coding, etc. sequence (usually)
- Usually defines phases or steps
- Often has notations
- Sometimes has tools

01-intro

28



## Waterfall Process Model

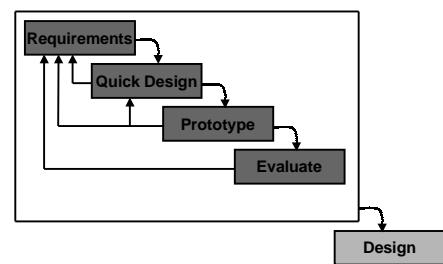


01-intro

29



## Prototyping Process Model



01-intro

30



## 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

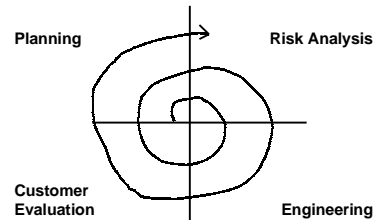
**NEVER Press a prototype into production**

01-intro

31



## Spiral Process Model



01-intro

32



## 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.96

01-intro

33



## 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

01-intro

34



## 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
    - ◆ 5 x cost during development
    - ◆ up to 100 x cost during maintenance
  - Hardware/software configuration
  - Security requirements
  - Real time requirements
  - Reliability requirements

01-intro

35



## 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

01-intro

36



## 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

01-intro

37



## 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

01-intro

38



## Systems Engineering

01-intro

39



## Computer System Engineering

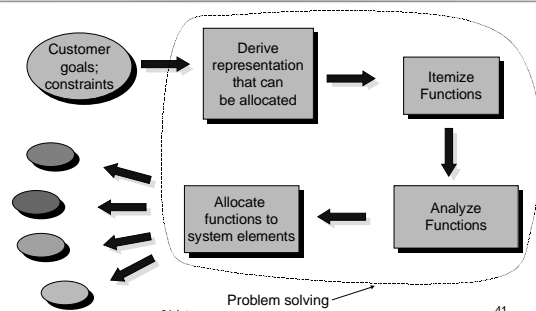
- Computer System Engineering is a problem-solving activity.
  - Itemize desired system functions
  - Analyze them
  - Allocate functions to individual system elements
- Systems Analyst (computer systems engineer)
  - Start with customer-defined goals and constraints
  - Derive a representation of function, 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.

01-intro

40



## Computer System Engineering



01-intro

41



## 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?)

01-intro

42



## Criteria for System Configuration: Business Issues

Criteria for allocation of function and performance to generic system elements:

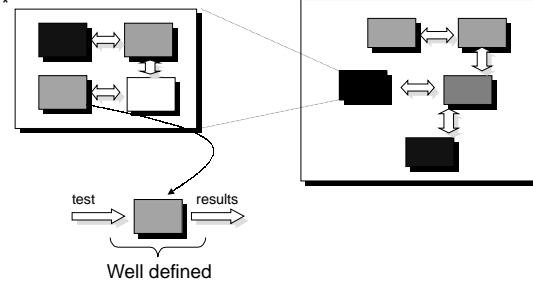
- Project\_Considerations: (cost, schedules, and risks)
- Business\_Considerations: (marketability, profitability)
- Legal\_Considerations: (liability, proprietary issues, infringement?)
- Human issues: (personnel trained? political problems, customer understanding of problem)

01-intro

43



## Hardware and Hardware Engineering



01-intro

44



## 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."

skip

01-intro

45



## Hardware Engineering

- Phases to system engineering of hardware:
  - Development Planning and requirements analysis:
    - ◆ best classes of hardware for problem,
    - ◆ availability of hardware
    - ◆ type of interface required
    - ◆ identification of what needs to be designed and built
  - Establish a Plan or "road map" for design implementation
    - ◆ May involve a hardware specification.
    - ◆ Use CAE/CAD to develop a prototype (breadboard)
    - ◆ Develop printed circuit (PC) boards
    - ◆ Manufacturing of boards

01-intro

46



## 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

01-intro

47



## Three high-level phases of Software Engineering

- Definition phase:
  - Software planning step → Software Project Plan
    - ◆ scope of project,
    - ◆ risk,
    - ◆ resource identification
    - ◆ cost and schedule estimates
  - Software Requirements Analysis → Requirements Specification
    - ◆ System element allocated to software is defined in detail.
    - ◆ Formal information domain analysis to establish models of information flow and structure (expand 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

01-intro

48



## 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 find maximum number of errors before shipping
  - Prepare software for release → Quality Assurance
  - Maintain software throughout its lifetime

01-intro

49



## 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.

01-intro

50



## Design Issues

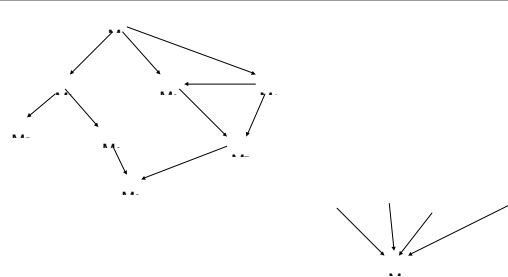
- Basic Design Principle:
  - Linguistic modular units: correspond to syntactic units in implementation language
  - Few interfaces: minimize the number of interfaces between modules
  - Small interfaces (weak coupling): minimize amount of info moving across interfaces
  - Explicit interfaces: when modules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

01-intro

51



## "Uses" Relation



01-intro

52



## 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*

01-intro

53



## Design Heuristics (cont'd)

- Define predictable functions for modules, but not too restrictive:
  - Black box modules are predictable (like hardware)
  - Restricting module processing to single subfunction (high cohesion)
  - Pass only a few (like 3, max) and return one parm from module
  - High maintenance: if randomly restrict local data structure size, options within control flow, or types of external interface
- "Single-entry-single-exit" modules: Avoid "Pathological Connections"
  - Enter at top, exit at bottom
  - Pathological connection: entry into middle of module
- Package SW based on design constraints and portability requirements
  - Assemble SW for specific processing environment
  - Program may "overlay" itself in memory reorganize group modules according to degree of repetition, access frequency, and interval of calls
  - Separate out modules only used once.

01-intro

54



## 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)

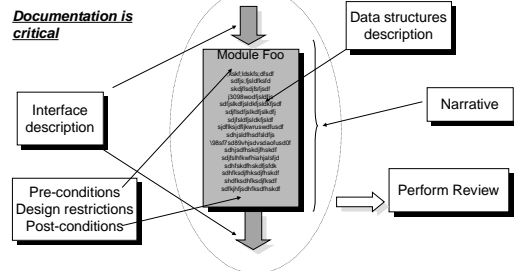
01-intro

55



## Design Postprocessing

***Documentation is critical***



01-intro

56



## 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)

01-intro

57