Welcome to ICS 121

Software Engineering =

study of software process,
development principles,
techniques and notations;

production of quality software,
delivered on time, within budget,
satisfying users' needs

- Administration, Syllabus
- Scope of Software Engineering
- Software Engineering Principles

Administration

- **Professor**
  - **David Redmiles**
- **Teaching Assistant**
  - **Jaya** Vaidyanathan
  - **Ian Lim**
- **Required**
  - Schach: Classical and Object-Oriented Software Engineering
  - Brooks: The Mythical Man-Month
  - Fowler: UML Distilled
  - occasional foundation papers, news clippings, etc.
- **Other References**
  - Ghezzi:Fundamentals of Software Engineering
  - Ian Sommerville: Software Engineering
- **Prerequisites**
  - Lower-division writing
  - ICS 52 (Grade C or better)
  - Math 6A(or ICS 6A)-B-C(or Math 3A)

Grading

- **Problem Analysis (5%)**
  - questions you need to have answered before continuing with project
- **Mockup (10%)**
  - end user scenario
- **Lifecycle Considerations and Validation (5%)**
  - anticipated changes, subset implementations, and validation plan
- **Requirements (15%)**
  - REBUS requirements specification
- **Design (15%)**
  - object-oriented
- **Midterm (15%)**
- **Final (20%)**
- **Homework (15%)**

Syllabus

- **Introduction to Software Engineering**
  - scope of Software Engineering
  - principles of Software Engineering
- **Software Nature and Qualities**
- **Software Production and Difficulties**
- **Software Lifecycle**
- **Lifecycle Validation and Testing Principles**
- **Requirements**
  - requirements process
  - requirements analysis and specification
  - system test plan
  - process
  - prototyping

Syllabus

- **Design**
  - general design process
  - design principles
  - integration test plan
  - design methods
    - » object-oriented
  - reuse

Syllabus

- **Formal (Module Interface) Specifications**
  - formal methods process
  - specification languages
    - » axiomatic specifications
    - » state machine specifications
    - » abstract model specifications
    - » algebraic specifications
  - module test plan
- **Software Testing, Verification and Validation**
  - verification vs. validation
  - testing process
  - unit testing
  - integration testing
  - system testing
  - verification and other analysis techniques
  - end user testing

Syllabus

- **Software Maintenance**
  - reverse- and re-engineering
- **Software Management and Planning**
  - scheduling and cost estimation
  - management structure and team organization
  - configuration management
- **Software Process Models**
- **Software Tools and Environments**

Software Engineering Scope

- **Software is typically delivered late, over budget, and faulty**
- **Software engineers require a broad range of skills applied to all phases of software production**
  - **Mathematics and Computer Science**
  - **Economics, Management, Psychology**
- **Scope of Software Engineering**
  - **Historical Aspects**
  - **Economic Aspects**
  - **Maintenance Aspects**
  - **Specification and Design Aspects**
  - **Team Programming Aspects**
  - **Verification and Validation Aspects**

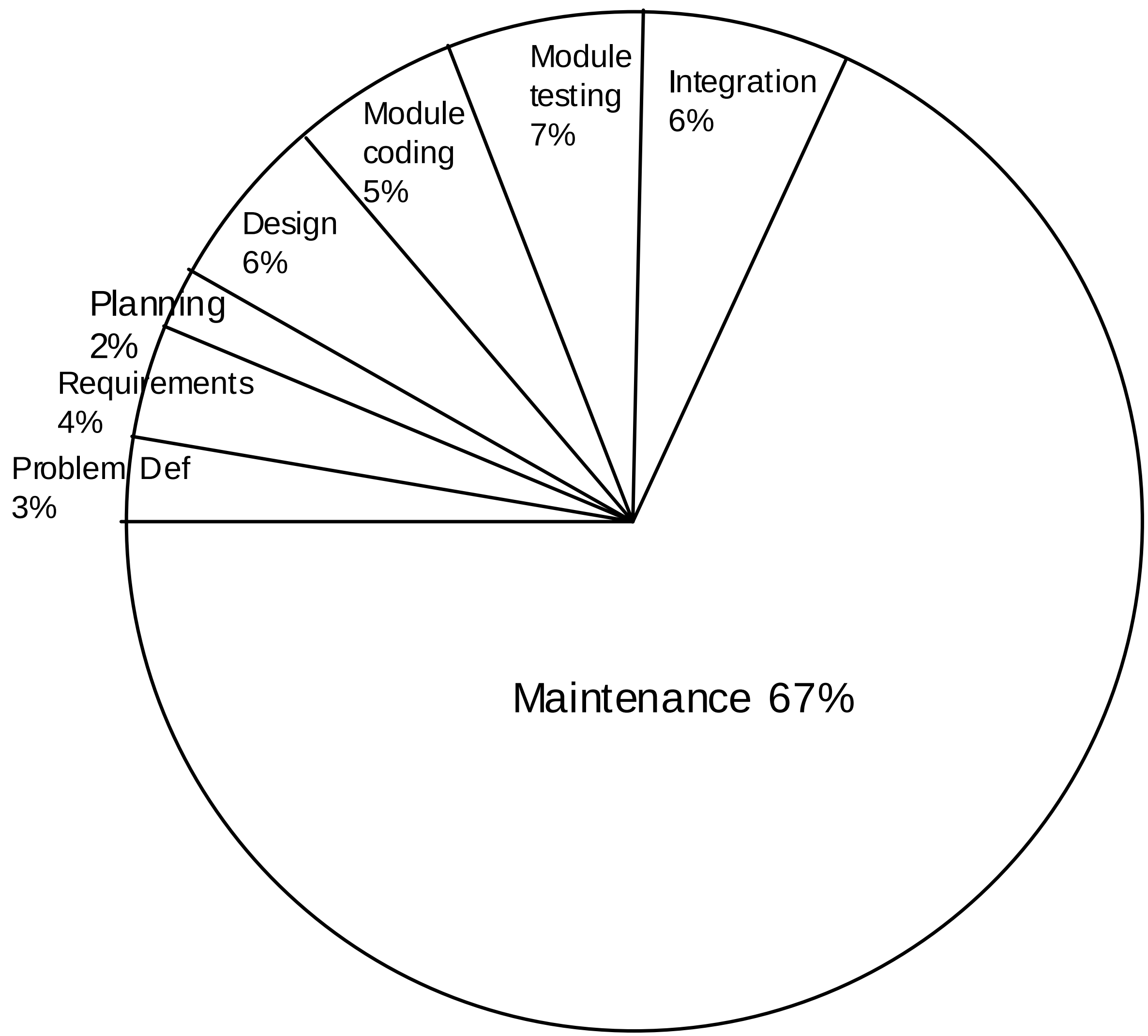
Historical Aspects

- NATO conference, 1968: coined term “software engineering”
  - software production should use established engineering principles to solve the software crisis
- DeRemer & Kron, 1976: PITL – “Programming In The Large”
- Parnas, 1987: “multi-person construction of multi-version software”
- Software engineering discipline is very young
  - techniques to specify properties of product independent of design are needed
  - formal analysis tools are critical
  - certain principles are essential
  - many techniques and notations

Economic and Maintenance Aspects

- Software Production = development + maintenance
- Quicker development is not always preferable
  - may lead to software that is difficult to maintain
  - resulting in higher long-term costs
- Maintenance costs are often over 50% of overall costs during the lifecycle of a software product
  - corrective maintenance (17.5%)
  - perfective maintenance (60.5%)
  - adaptive maintenance (18%)
- Real world is constantly changing
  - all software products undergo maintenance to account for change

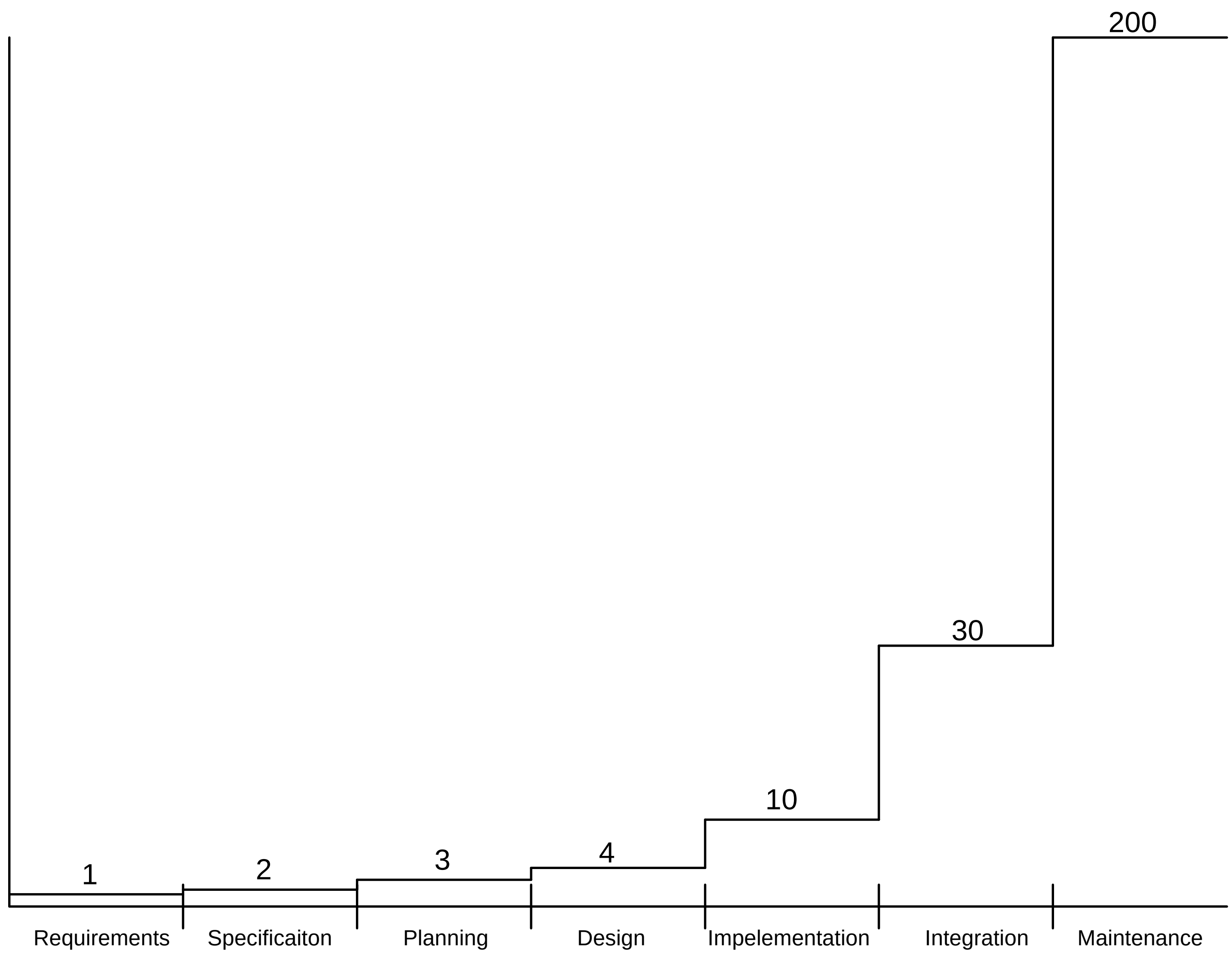
Maintenance Costs



Requirements and Design Aspects Verification and Validation Aspects

- **The longer a fault exists in software**
  - the more costly it is to detect and correct
  - the less likely it is to be fixed correctly
- **60-70% of all faults detected in large-scale software projects are introduced in requirements and design**
- **Faults must be found early**
  - faults must be found early through specification and design validation
- **Verification and validation must be done throughout the lifecycle**
  - validate first description
  - verify current phase with respect to previous
  - evaluate testability at each phase
  - develop test plans based on each phase

Specification and Design Aspects relative cost of fixing an fault

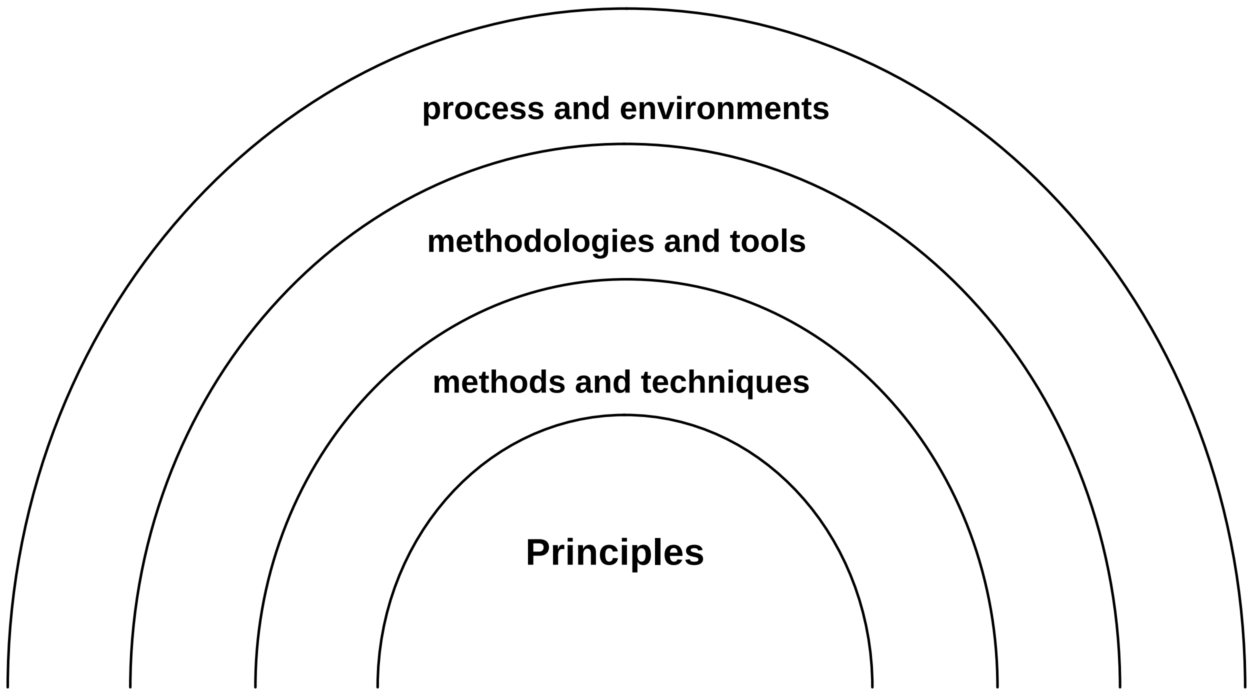


Team Programming Aspects

- **Reduced hardware costs affords hardware that can run large products – products too large for an individual to develop**
- **Most software is produced by a team of software engineers, not an individual**
  - **Team programming leads to interface problem between components and communications problems between members**
  - **Team programming requires good team organization to avoid excessive conferences**

Software Engineering Principles

- Deal with both process and product
- Applicable throughout lifecycle
- Need abstract descriptions of desirable properties
- Same principles as other engineering disciplines



Rigor and Formality

- Rigor is a necessary complement to creativity
- Rigor enhances understandability, improves reliability, facilitates assessment, and controls cost
- Formality is the highest degree of rigor
  - mathematically defined
- Engineering = sequence of well-defined, precisely-stated, sound steps, which follow method or apply technique based on some combination of
  - theoretical results derived from formal model
  - empirical adjustments for unmodeled phenomenon
  - rules of thumb based on experience

Separation of Concerns

- **Enables mastering of inherent complexity**
- **Allows concentration on individual aspects**
  - **product features: functions, reliability, efficiency, environment, user interface, etc.**
  - **process features: development environment, team organization, scheduling, methods,**
  - **economics and management**
- **Concerns may be separated by**
  - **time (process sequence)**
  - **qualities (e.g., correctness vs. performance)**
  - **views to be analyzed separately (data vs. control)**
  - **components**
- **Leads to separation of responsibility**

Modularity and Decomposition

- **Complex system divided into modules**
- **Modular decomposition allows separation of concerns in two phases**

bottom-up ↓ **aspects of modules in isolation** ↑ top-down

overall characteristics of integrated system

- **Modularity manages complexity, fosters reusability, and enhances understandability**
  - **composibility vs. decomposibility**
  - **high cohesion and low coupling**

Abstraction

- **Identify important aspects and ignore details**
- **Abstraction depends on the purpose or view**
- **Models are abstractions of reality**
- **Abstraction permeates software development**
  - **from requirements to code**
  - **from natural language descriptions to mathematical models**
  - **from products to process**
- **One specification but many realizations**

Anticipation of Change

- **Constant change is inevitable in large-scale software systems**
  - software repair & error elimination
  - evolution of the application
- **Identify likely changes and plan for change**
  - software requirements usually not entirely understood
  - users and environments change
  - also affects management of software process
- **Maintenance is process of error correction and modification to reflect changing requirements**
  - regression testing with maintenance
  - configuration management of versions

Generality

- **Focus on discoving more general problem than the one at hand**
  - fosters potential reuse
  - facilitates identification of OTS solution
- **Trade-offs between initial costs vs. reuse savings**
- **General-purpose, OTS products are general trend in application domains**
  - standard solutions to common problems

Incrementality

- Step-wise process with successively closer approximations to desired goal
- Identify and "deliver" early subsets to gain early feedback
  - fosters controlled evolution
- Incremental concentration on required qualities
- Intermediate deliverables may be prototypes
- Requires careful configuration management and documentation

Reliability

- **As software application pervades critical systems, reliability is paramount**
- **Cost of failure exceeds cost of development**
- **Reliability measures how well a system provides expected service over time**
  - all service is not equal
  - software reliability based entirely on development
  - software does not degrade

Formal development methods lead to higher reliability

Formal analysis techniques are critical

Relationships between Principles
Discussion

- formality and modularity
- formality and separation of concerns
- separation of concerns and modularity
- modularity and abstraction
- modularity and anticipation of change
- anticipation of change and generality
- abstraction and generality
- modularity and incrementality
- anticipation of change and incrementality
- generality and incrementality