Software’s Dual Role

- Software is a product
  - Delivers computing potential
  - Produces, manages, acquires, modifies, displays, or transmits information
- Software is a vehicle for delivering a product
  - Supports or directly provides system functionality
  - Controls other programs (e.g., an operating system)
  - Effects communications (e.g., networking software)
  - Helps build other software (e.g., software tools)

Wear vs. Deterioration

- Idealized curve
- Actual curve
- Time
- Failure rate
- Increased failure rate due to side effects
- Change
- Idealized curve
- Actual curve
Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth, but …
  - Invariably lead to bad decisions, therefore …
  - Insist on reality as you navigate your way through software engineering

A Layered Technology

Software Engineering

- tools
- methods
- process model
- quality goals
Framework Activities

- Communication
- Planning
- Modeling
- Analysis of requirements
- Design
- Construction
- Testing
- Deployment

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

The Waterfall Model

[Diagram of the Waterfall Model with steps: project initiation, requirement gathering, analysis, design, code, testing, deployment, feedback, tracking, scheduling.]
**Evolutionary Models: Prototyping**

- Quick plan
- Construction
- Prototype model
- Communication
- Quick design
- Delivery & Feedback
- Deployment

---

**The Manifesto for Agile Software Development**

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Kent Beck et al

---

**An Agile Process**

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur
Extreme Programming (XP)

- User stories
- Acceptance test
- Design
- Pair programming
- Continuous integration
- Test
- Iteration plan
- Refactoring
- Simple design
- CRC cards
- Prototypes
- Spike solutions
- Project velocity computed
- User stories validated
- Acceptance test criteria
- Iteration plan
- Pair programming
- Continuous integration
- Test
- Simple design
- CRC cards
- Prototypes
- Spike solutions
- Project velocity computed

Core Software Engineering Principles

- Provide value to the customer and the user
- KIS—keep it simple!
- Maintain the product and project “vision”
- What you produce, others will consume
- Be open to the future
- Plan ahead for reuse
- Think!

System Engineering

- Elements of a computer-based system
  - Software
  - Hardware
  - People
  - Dataflow
  - Documentation
  - Procedures
- Systems
  - A hierarchy of macro-elements
The Hierarchy

System Modeling

Requirements Engineering-II
Analysis Model -> Design Model

Design Principles

Fundamental Concepts

- abstraction—data, procedure, control
- architecture—the overall structure of the software
- patterns—"conveys the essence" of a proven design solution
- modularity—compartmentalization of data and function
- hiding—controlled interfaces
- Functional independence—single-minded function and low coupling
- Refactoring—a reorganization technique that simplifies the design
Testing Strategy

- We begin by “testing-in-the-small” and move toward “testing-in-the-large”
- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows
- For OO software
  - Our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

Strategic Issues

- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes “rapid cycle testing.”
- Build “robust” software that is designed to test itself.
- Use effective formal technical reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

Testability

- Operability—it operates clearly
- Observability—the results of each test case are readily observed
- Controllability—the degree to which testing can be automated and optimized
- Decomposability—testing can be targeted
- Simplicity—reduce complex architecture and logic to simplify tests
- Stability—few changes are requested during testing
- Understandability—of the design
Exhaustive Testing

There are $10^6$ possible paths! If we execute one test per millisecond, it would take 3,170 years to test this program!!

Boundary Value Analysis

input domain output domain

OOT Methods: Partition Testing

- Partition Testing
  - reduces the number of test cases required to test a class in much the same way as equivalence partitioning for conventional software
  - state-based partitioning
    - categorize and test operations based on their ability to change the state of a class
  - attribute-based partitioning
    - categorize and test operations based on the attributes that they use
  - category-based partitioning
    - categorize and test operations based on the generic function each performs
**Measures, Metrics and Indicators**

- A measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.
- The IEEE glossary defines a **metric** as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute.”
- An **indicator** is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself.

**Measurement Principles**

- The objectives of measurement should be established before data collection begins.
- Each technical metric should be defined in an unambiguous manner.
- Metrics should be derived based on a theory that is valid for the domain of application (e.g., metrics for design should draw upon basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable).
- Metrics should be tailored to best accommodate specific products and processes [BAS84].

**Goal-Oriented Software Measurement**

- The **Goal/Question/Metric Paradigm**
  1. Establish an explicit measurement goal that is specific to the process activity or product characteristic that is to be assessed.
  2. Define a set of questions that must be answered in order to achieve the goal.
  3. Identify well-formulated metrics that help to answer these questions.

**Goal definition template**

- **Analyze** (the name of activity or attribute to be measured)
  - for the purpose of (the overall objective of the analysis)
  - with respect to (the aspect of the activity or attribute that is considered)
  - from the viewpoint of (the people who have an interest in the measurement)
  - in the context of (the environment in which the measurement takes place).
Software Teams

How to lead?
How to collaborate?
How to motivate?
How to organize?
How to create good ideas?

The Project

- Projects get into trouble when …
  - Software people don’t understand their customer’s needs.
  - The product scope is poorly defined.
  - Changes are managed poorly.
  - The chosen technology changes.
  - Business needs change (or are ill-defined).
  - Deadlines are unrealistic.
  - Users are resistant.
  - Sponsorship is lost (or was never properly obtained).
  - The project team lacks people with appropriate skills.
  - Managers (and practitioners) avoid best practices and lessons learned.

A Good Manager Measures

What do we use as a basis?
- size?
- function?

measurement

process

process metrics

project metrics

product

product metrics
Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
- Risk analysis is considered in detail in Chapter 25.
  - Define required solutions
    - Define functional requirements
    - Define non-functional requirements
    - Identify environmental resources

Project Planning Task Set-II

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use cases
  - Reconcile the estimates

- Develop a project schedule
  - Scheduling is considered in detail in Chapter 24.
    - Establish a meaningful task set
    - Define a task network
    - Use scheduling tools to develop a timeline chart
    - Define schedule tracking mechanisms

Scheduling Principles

- compartmentalization—define distinct tasks
- interdependency—indicate task interrelationship
- effort validation—be sure resources are available
- defined responsibilities—people must be assigned
- defined outcomes—each task must have an output
- defined milestones—review for quality
Timeline Charts

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Week 1</th>
<th>Week 2</th>
<th>Week 3</th>
<th>Week 4</th>
<th>Week n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Software Quality Assurance

- Process Definition & Standards
- Formal Technical Reviews
- Analysis & Reporting
- Measurement
- Test Planning & Review

Importance of Software-Revisited

- In Chapter 1, software was characterized as a differentiator.
  - The function delivered by software differentiates products, systems, and services and provides competitive advantage in the marketplace.
- But software is more than a differentiator.
  - The programs, documents, and data that are software help to generate the most important commodity that any individual, business, or government can acquire—information.
Software Engineering Ethics-I


- Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.
8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.