SOFTWARE ENGINEERING AND PROJECT MANAGEMENT

MD. MAHADI HASAN SHAON

SOFTWARE ENGINEERING AND PROJECT MANAGEMENT

Course Code:	CSE-306	Credits:	03
		CIE Marks:	90
Exam Hours:	03	SEE Marks:	60

Course Learning Outcome (CLOs): After Completing this course successfully, the student will be able to…

CLO 1	Describe the fundamental principles of software engineering and project management, including software development life cycle models, project management methodologies, and software quality assurance techniques.
CLO 2	Understand and analyze the requirements, design, and implementation phases of software development projects using appropriate software engineering tools and techniques.
CLO 3	Create and implement project plans, schedules, and budgets for software development projects, and effectively manage project resources and risks.
CLO 4	Apply knowledge of software engineering and project management concepts to identify, evaluate, and propose solutions to software development challenges and issues.
CLO 5	Identify and explain the ethical and professional responsibilities of software engineers and project managers, and apply ethical principles to software development projects.

SUMMARY OF COURSE CONTENT

Serial No.	SUMMARY OF COURSE CONTENT	Hours	CLOs
1	Software Development Life Cycle: This topic will cover the different phases of software development, including requirements gathering, design, implementation, testing, deployment, and maintenance.	8	CLO1 CLO2
2	Agile Development: This topic will introduce agile methodologies for software development, including Scrum, Kanban, and Extreme Programming. It will cover topics such as user stories, sprints, backlog management, and continuous integration.	8	CLO1 CLO2
3	Project Management: This topic will introduce project management techniques, including project planning, risk management, change management, and resource allocation. It will also cover project monitoring and control.	8	CLO1 CLO2
4	Software Testing: This topic will cover different types of software testing, including unit testing, integration testing, system testing, and acceptance testing. It will also cover tools and techniques for test automation.	8	CLO4 CLO5
5	Software Quality Assurance: This topic will cover different techniques for ensuring software quality, including code reviews, static analysis, dynamic analysis, and quality metrics. It will also cover techniques for measuring and improving software maintainability and reliability.	8	CLO5

Recommended Books:

- Roger S. Pressmann, "Software Engineering - A Practitioner's Approach", McGraw-Hill, latest Edition
- Reference:
  Ian Sommerville, "Software Engineering", Pearson Education, latest Edition

ASSESSMENT PATTERN

CIE- Continuous Internal Evaluation (90 Marks)

Bloom's Category Marks (out of 90)	Tests (45)	Assignments (15)	Quizzes (15)	Attendance (15)
Remember	5	03		
Understand	5	04	05	
Apply	15	05	05	
Analyze	10			
Evaluate	5	03	05	
Create	5			

SEE- Semester End Examination (60 Marks)

Bloom's Category	Test
Remember	7
Understand	7
Apply	20
Analyze	15
Evaluate	6
Create	5

Week No.	Topics	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
1	• Introduction to Software Engineering • Describe SE concepts, history, and applications.	Lecture, multimedia, group discussion	Feedback, Q&A, assessment of LOs	CLO1
2	• Software Process Models • Explain and compare traditional and agile process models.	Lecture, multimedia, interactive sessions	Feedback, Q&A, quizzes	CLO2
3	• Requirements Engineering • Apply requirements elicitation and specification techniques.	Lecture, multimedia, hands-on practice	Mid term Quiz, assessment of LOs	CLO3
4	• System Design and Architecture • Discuss and apply design principles and patterns.	Lecture, multimedia, interactive sessions	Feedback, Q&A, assessment of LOs	CLO4
5	• Software Development • Implement coding standards and best practices.	Lecture, multimedia, practical examples	Midterm Case Study, Home Assignment	CLO5
6	• Software Testing • Explain and apply various testing strategies and tools.	Lecture, multimedia, interactive sessions	Feedback, Q&A, quizzes, group discussions	CLO6
7	• Software Project Management • Manage software projects including planning, estimation, and scheduling.	Lecture, multimedia, group work	Feedback, Q&A, assessment of LOs	CLO7
8	• Software Configuration Management • Use version control systems and manage software configurations.	Lecture, multimedia, hands-on practice	Mid term Quiz , Home Assignment	CLO8

Week No.	Topics	Teaching Learning Strategy(s)	Assessment Strategy(s)	Alignment to CLO
9	• Software Metrics and Measurement • Discuss and apply software metrics and measurement techniques.	Lecture, multimedia, interactive sessions	Feedback, Q&A, assessment of LOs	CLO9
10	• Software Maintenance and Evolution • Understand maintenance processes and challenges.	Lecture, multimedia, problem-solving sessions	Feedback, Q&A, group discussions	CLO9
11	• Software Engineering Ethics • Discuss ethical issues and professional responsibilities.	Lecture, multimedia, interactive sessions	Final term Quiz, assessment of LOs	CLO3
12	• Advanced Topics: Agile Development • Explore advanced topics and emerging trends in software engineering.	Lecture, multimedia, practical examples	Feedback, Q&A, group discussions.	CLO10
13	• Software Security • Principles of Agile methodologies. • Key frameworks: Scrum, Kanban, and XP.	Lecture, multimedia, practical examples	Feedback, Q&A, group discussions	CLO9
14	• Emerging Trends in Software Engineering • AI and machine learning in software development.	Lecture, multimedia, practical examples	Feedback, Q&A, quizzes, group discussions	CLO6
15	• Introduction to Object-Oriented Analysis • Introduction to OOA. • Basic user requirements must be communicated between the customer and the software engineer, • Classes must be identified.	Lecture, multimedia, group discussion	Feedback, Q&A, assessment of LOs	CLO1
16	• Quality Assurance • Tools for static and dynamic analysis.	Lecture, multimedia, practical examples	Final term Case Study #1, Home Assignment #1	CLO5
17	• Final Project Presentations • Students present their projects demonstrating the application of course concepts.	Lecture, multimedia, practical examples	Final Term	CLO11

WEEK 1

WHAT IS SW ENGINEERING ABOUT?

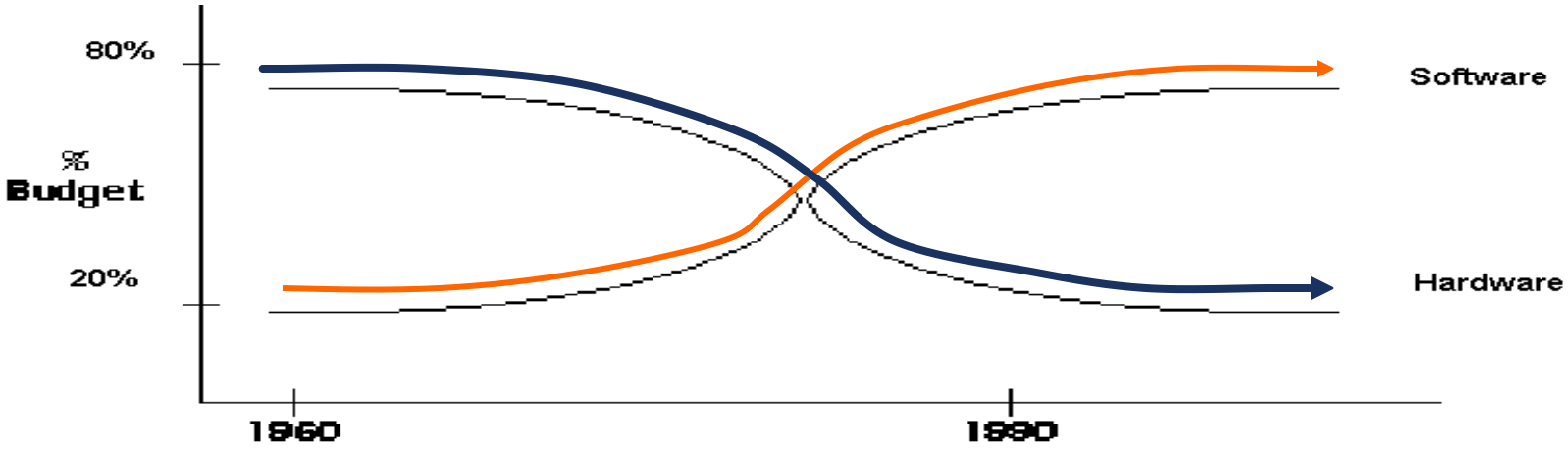
- Project Management
- Configuration Management
- Structured Methodology
- Object Oriented Analysis / Object Oriented Design
- User Interface Design
- Testing and Validation
- Quality Assurance

WHY IS SOFTWARE ENGINEERING IMPORTANT?

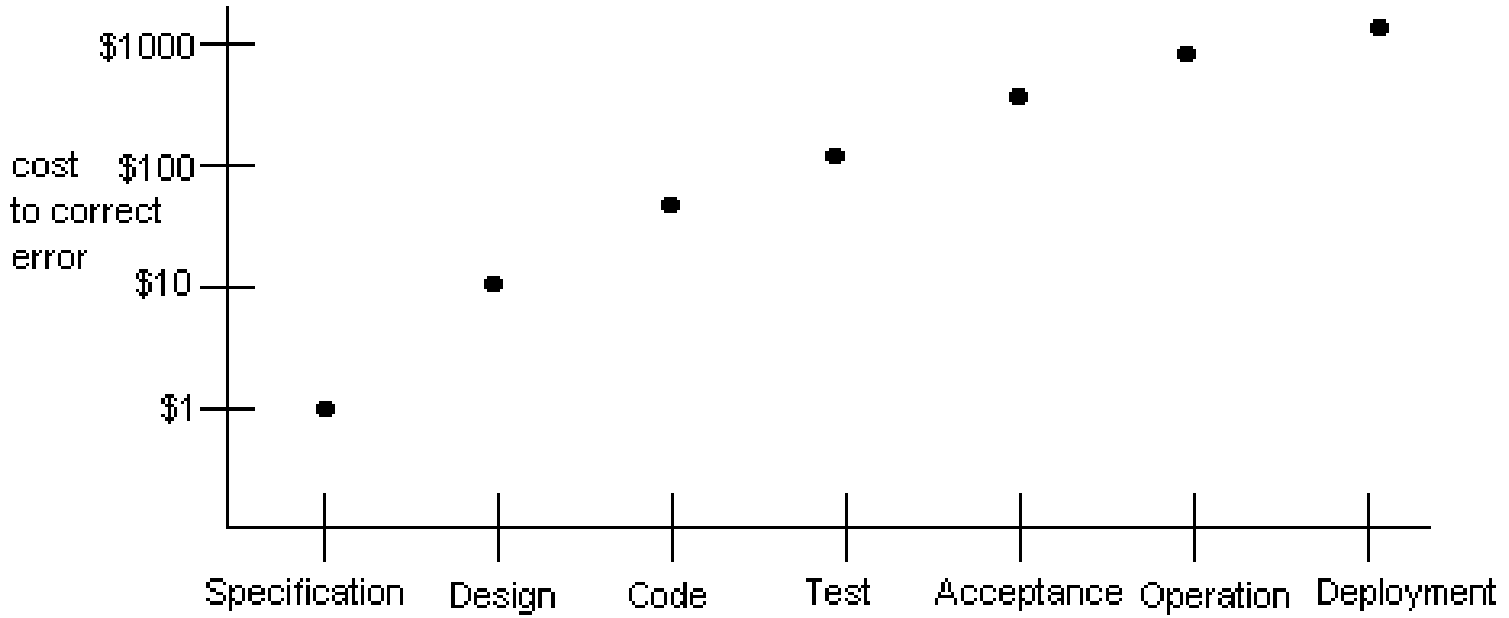
To avoid costly errors caused by software:

- Lost Voyager Spacecraft (one bad line of code caused failure)
- Several people killed by a radiation machine (no means of catching operator errors)
- Commercial aircraft accidentally shot down during Gulf War (poor user interface)
- Toma Hawk missile defense system cause more damage to Israel than Scud - non-existence of overflow detection was the source of the problem

HISTORICAL PROJECT COST ALLOCATION



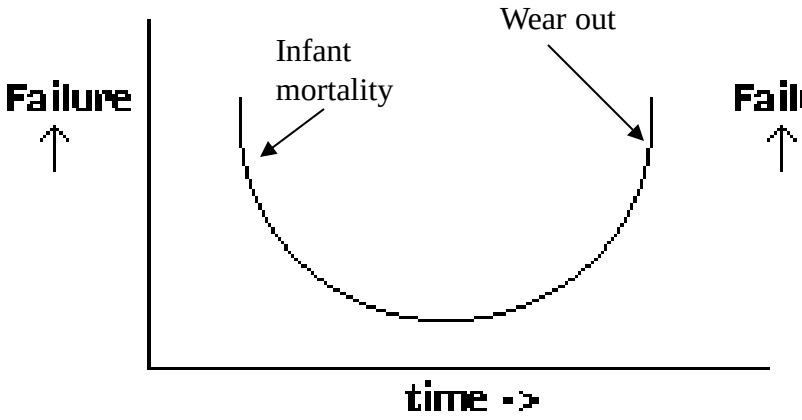
EARLY ERROR DETECTION SAVES MONEY



SOFTWARE CHARACTERISTICS

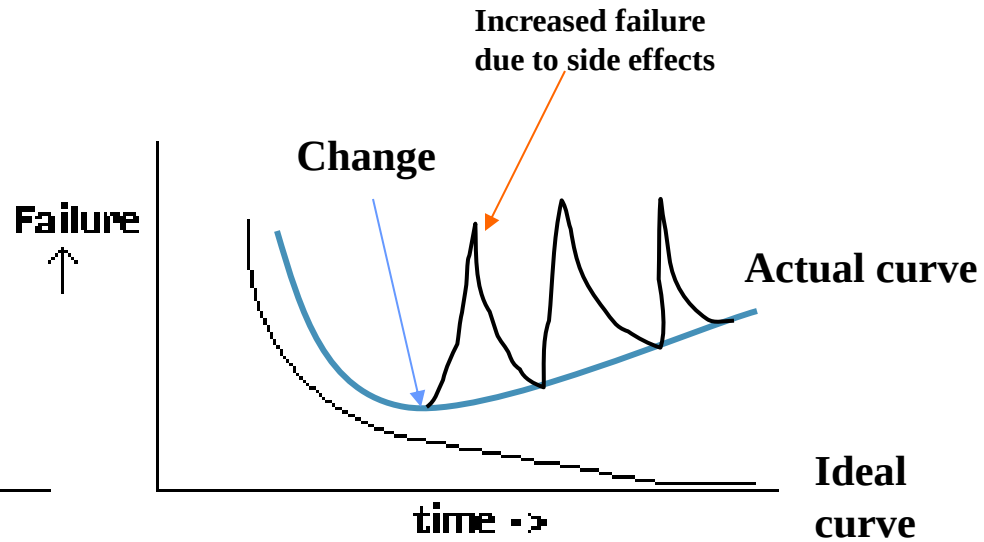
- Software is both a product and a vehicle for developing a product.
- Software is engineered not manufactured.
- Software does not wear out, but it does deteriorate.
- Although the industry is moving toward component based construction, most software is still custom-built.

FAILURES OVER TIME



Hardware

(Bathtub curve)



Software

SOFTWARE CRISIS

- Software failures receive a lot more publicity than software engineering success stories.
- The problems that afflict software development are more likely to be associated with how to develop and support software properly, than with simply building software that functions correctly.

SOFTWARE MYTHS – PART 1

- Software standards provide software engineers with all the guidance they need - Learn to use them
- People with modern computers have all the software development tools they need - Need good CASE tools
- Adding people is a good way to catch up when a project is behind schedule - Late addition makes it later
- Giving software projects to outside parties to develop solves software project management problems - Need to learn how to manage and control software projects

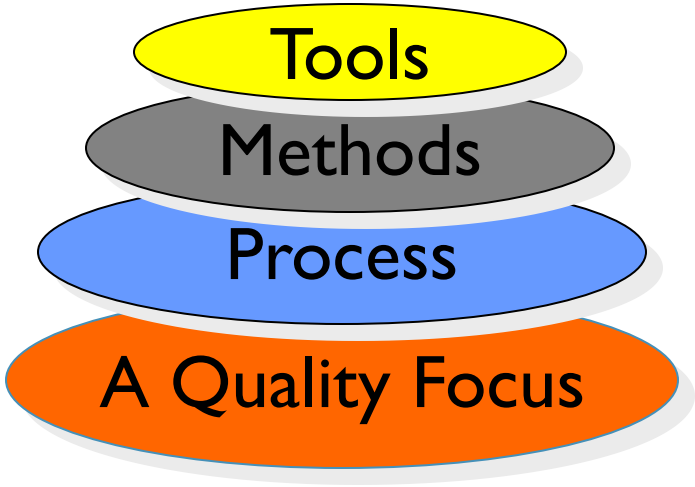
SOFTWARE MYTHS – PART 2

- A general statement of objectives from the customer is all that is needed to begin a software project - spend time to have very good understanding of customer requirements
- Project requirements change continually and change is easy to accommodate in the software design
  - Understand the requirements first then write codes. It costs more to change later
- Once a program is written, the software engineer's work is finished
  - It is only the beginning

SOFTWARE MYTHS – PART 3

- There is no way to assess the quality of a piece of software until it is actually running on some machine. The only deliverable from a successful software project is the working program.
  - Practice formal/peer review
- Software engineering is all about the creation of large and unnecessary documentation not shorter development times or reduced costs
  - Better quality leads to reduced work

- A strategy for producing high quality software.
- It is a layered technology
- Software engineering encompasses a process, management techniques, technical methods, and the use of tools
- The bedrock is the quality focus



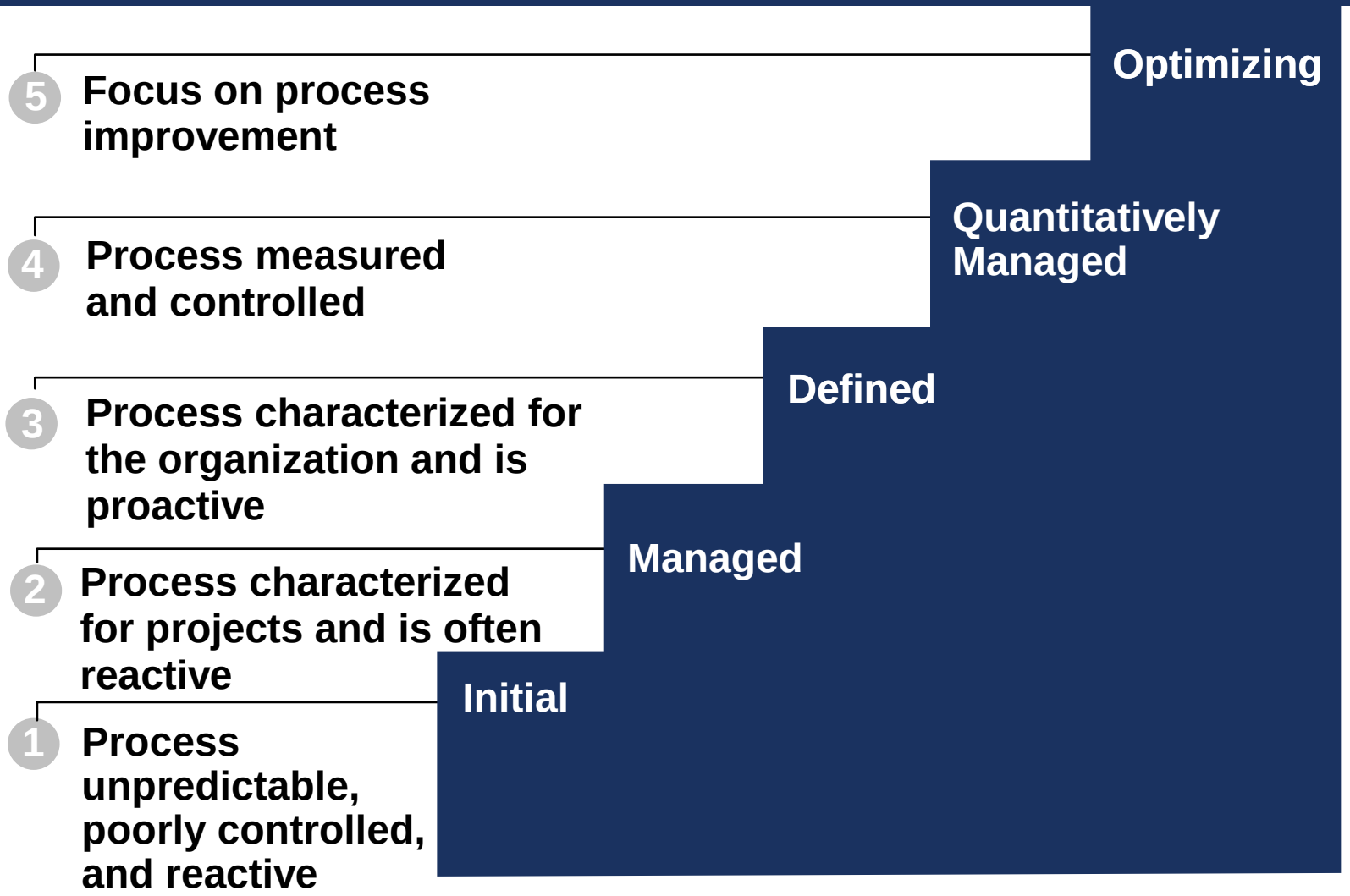
SOFTWARE ENGINEERING

- Process - foundation
  - It defines the framework that must be established for effective delivery of software engineering technology
  - A framework for a set of key process areas (KPA)
  - KPA form the basis for management control of software projects
- Methods - technical how-to's for building software
  - requirements analysis
  - design
  - program construction, testing and support
- Tools
  - Automation/semi-automation
  - CASE/CAD/CAE tools

WHAT IS CMMI?

- Capability Maturity Model Integration (CMMI)
- The CMMI model is a collection of best practices from leading engineering companies.
- It is a Software Engineering Institute's (SEI) comprehensive process meta-model
- It has two different representations:
  - *Continuous* - uses capability levels to measure process improvement
  - *Staged* - uses maturity levels to measure organization's overall maturity

THE MATURITY LEVELS (STAGED)



THE CAPABILITY LEVELS (CONTINUOUS)

WEEK 2

COMMON PROCESS FRAMEWORK

- Framework activities - *applicable to all software projects*
  - Customer communication
  - Planning
  - Risk analysis
  - Engineering
  - Construction and release
  - Customer evaluation
- Umbrella activities - *applicable across entire software process*
  - Project tracking and control
  - Technical review
  - Quality assurance
  - Risk management
  - Software configuration management
  - Measurements
  - Metrics collection

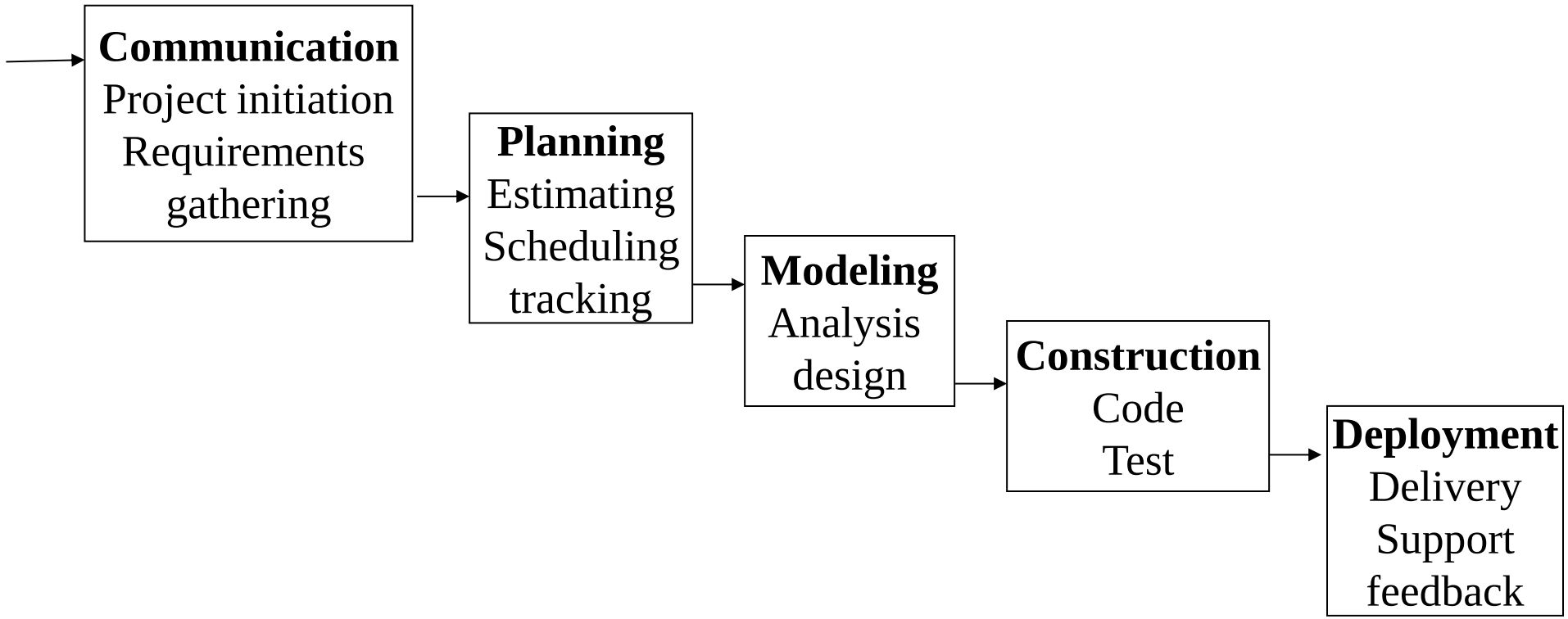
SOFTWARE LIFE CYCLE PHASES

- Requirements, analysis, and design phase.
- System design phase.
- Program design phase.
- Program implementation phase.
- Unit testing phase.
- Integration testing phase.
- System testing phase.
- System delivery.
- Maintenance.

GENERIC SOFTWARE PROCESS MODELS

- The waterfall or linear sequential model
  - Separate and distinct phases of specification and development
- Prototype model
  - Requirements are not clear
- Rapid Application Development (RAD) model
  - High speed adaptation of linear sequential model

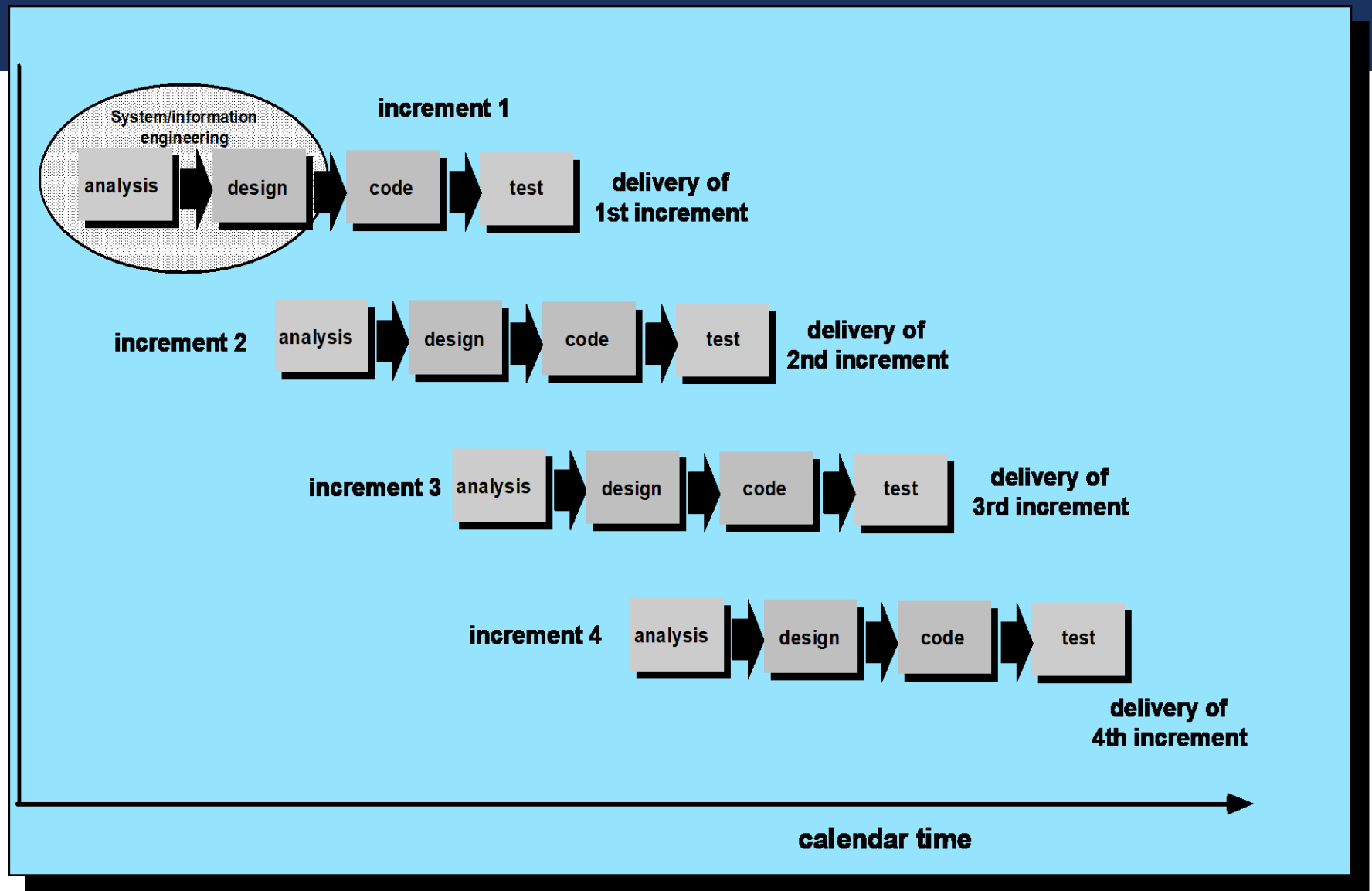
THE LINEAR/WATERFALL MODEL



WATERFALL MODEL PROBLEMS

- This model assumes that requirements are well understood
- Inflexible partitioning makes it difficult to respond to changing customer requirements
- It is often difficult for the customer to state all the requirements explicitly
- A working version of the program will not be available until late in the project
- Example: A complete Java editor delivered at the end of project

THE INCREMENTAL MODEL



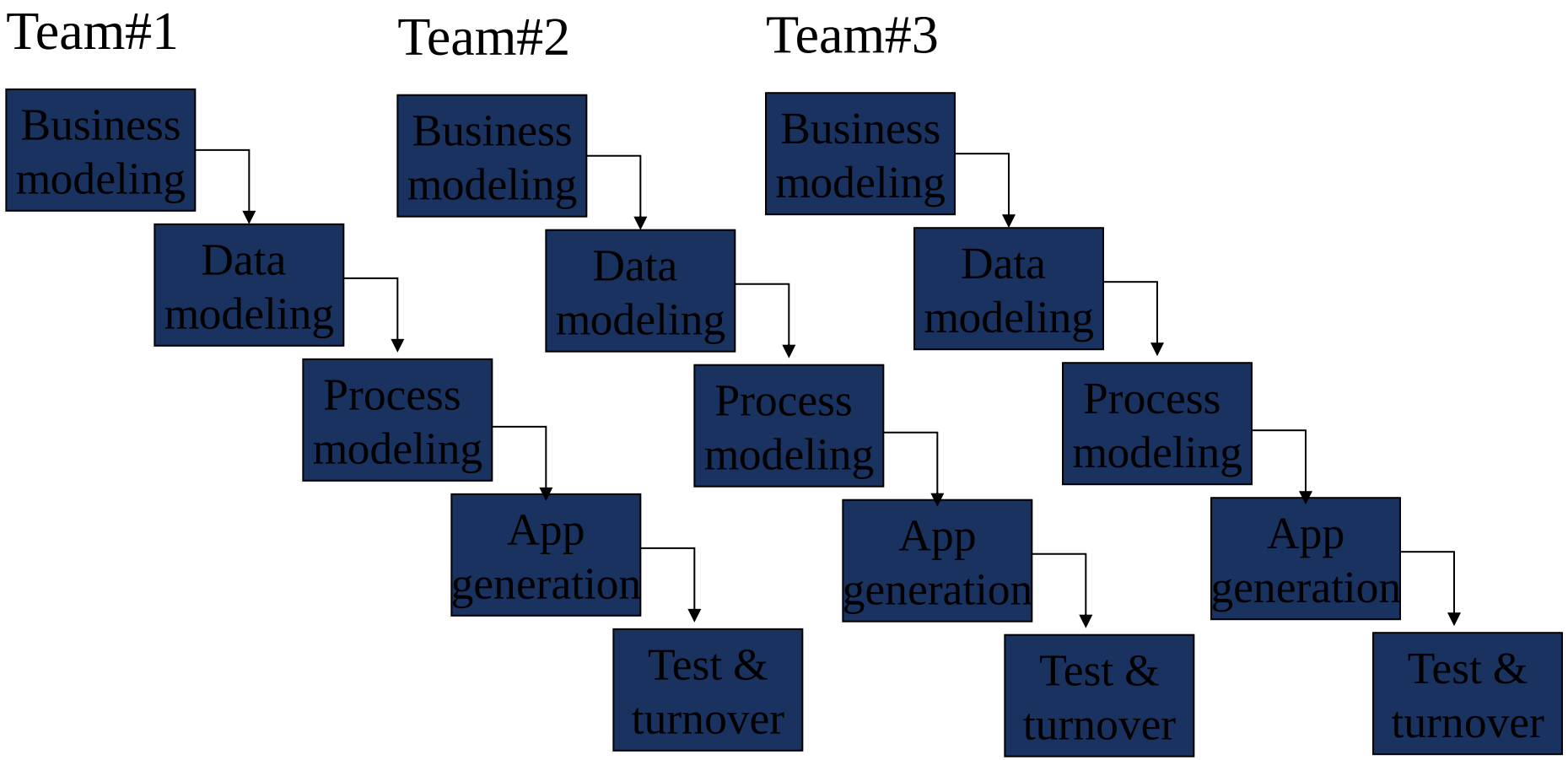
INCREMENTAL DEVELOPMENT ADVANTAGES

- Deliver the core product first
- Add on / refine features
- Provide a platform for evaluation by user
- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing
- It is particularly useful when staffing is unavailable for a complete implementation

RAPID APPLICATION DEVELOPMENT (RAD) MODEL

- Characteristics:
  - Incremental process model with extremely short development cycle
  - High speed adaptation of Waterfall model where rapid development is achieved by component based construction approach i.e. develop a small component rather than a system
  - Used for information system applications
- Drawbacks:
  - Need more resource for large projects
  - Need commitment from all parties
  - Not suitable for high performance and when technical risk is high

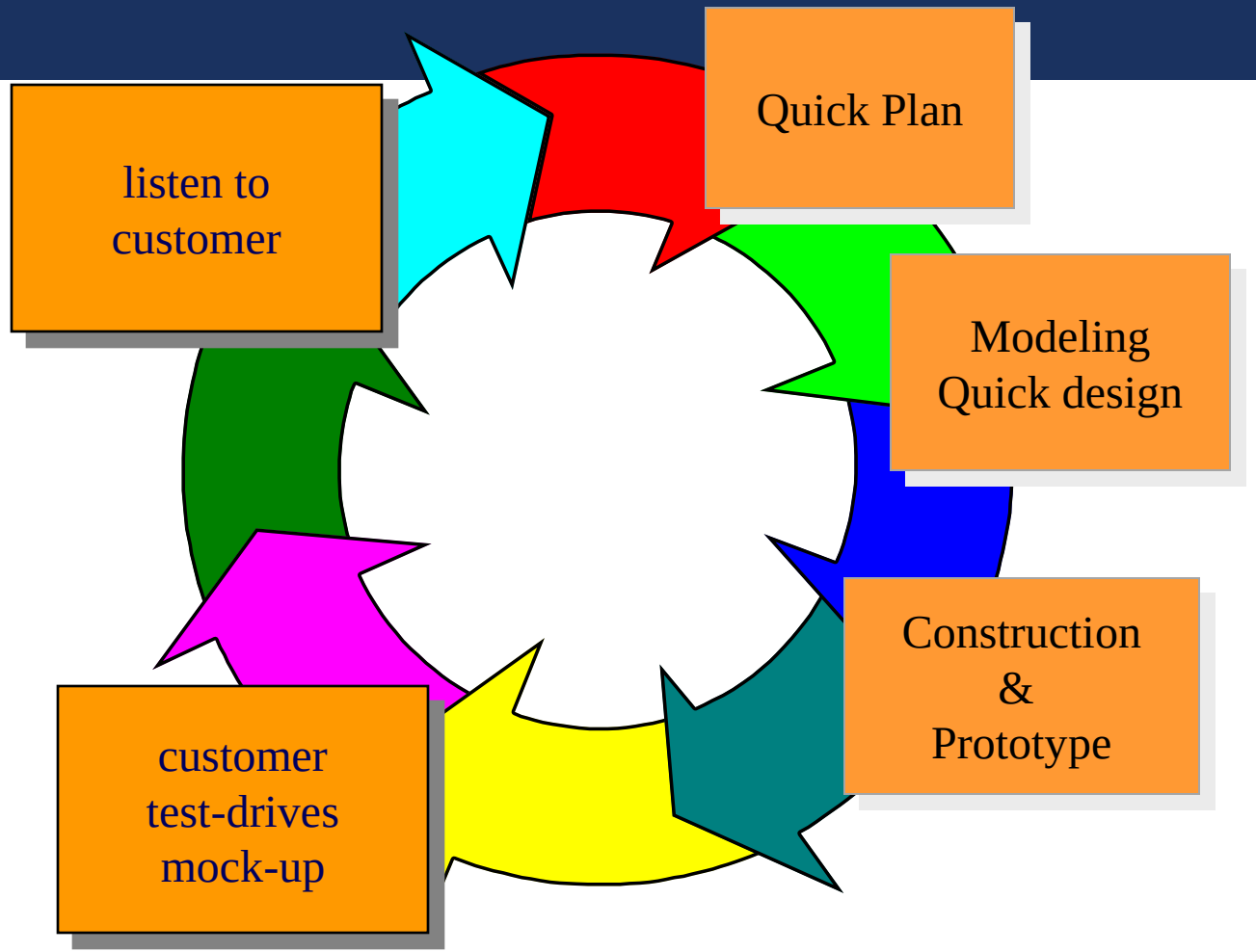
RAD MODEL


Team#1
Team#2
Team#3
Business modeling
Data modeling
Process modeling
App generation
Test & turnover
Business modeling
Data modeling
Process modeling
App generation
Test & turnover
Business modeling
Data modeling
Process modeling
App generation
Test & turnover

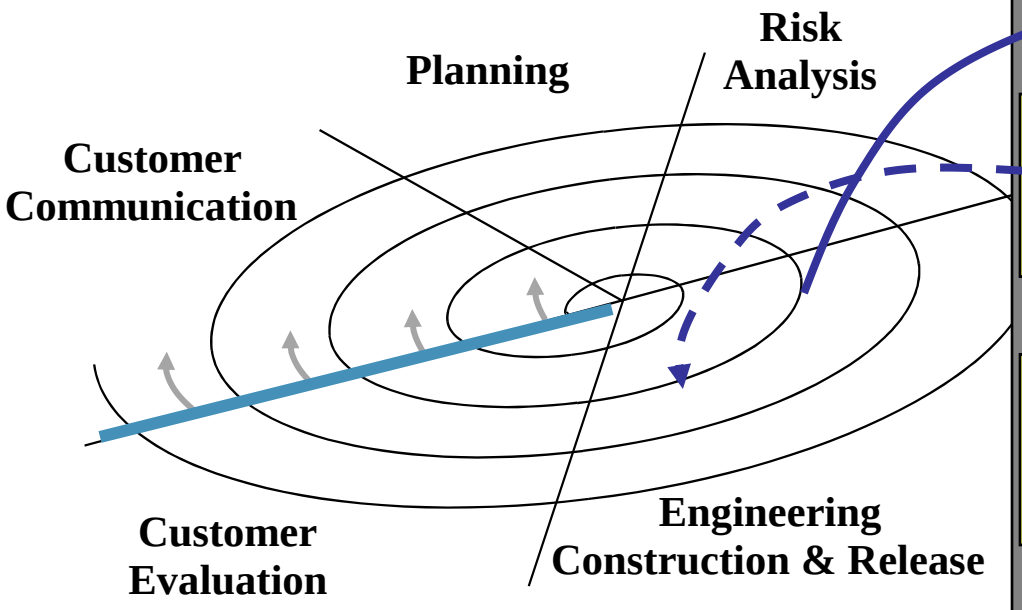
PROTOTYPING

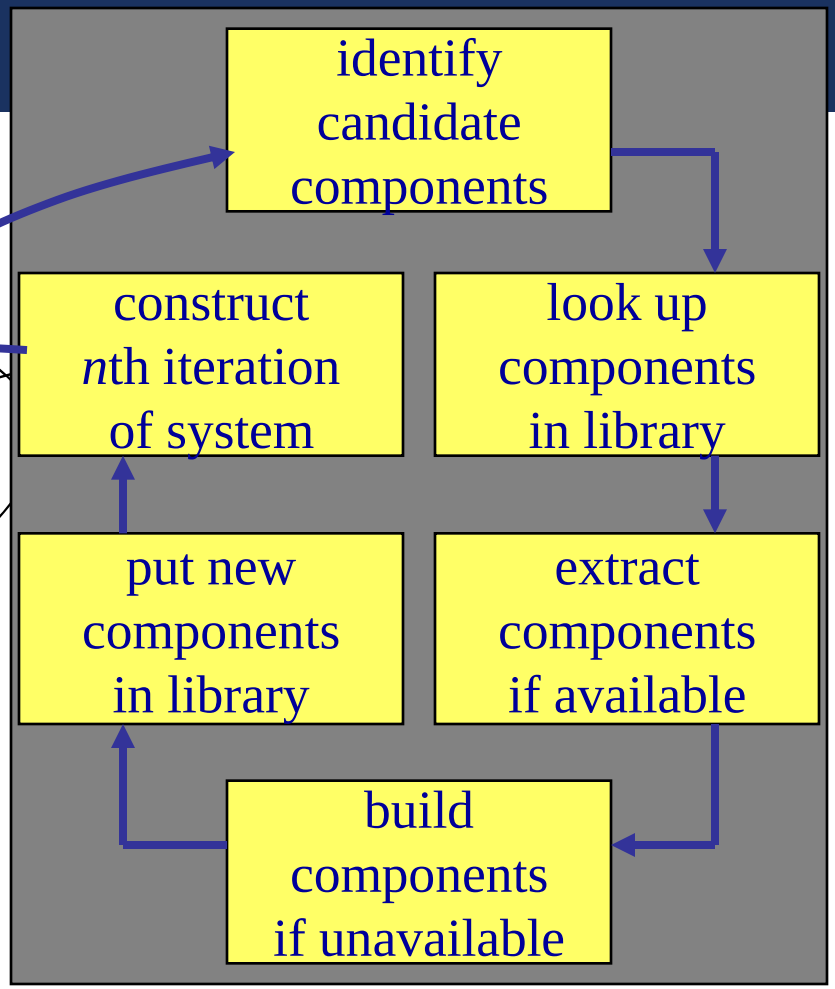
- It is an evolutionary model
- Customer often defines general objectives not detailed input/output and processing requirements
- Being unsure about
  - efficiency of an algorithm
  - human-computer interaction
- Often used for identifying requirements
  - Throw-away prototype (Risk?)
- Quick planning and modeling is used
- Problems:
  - Customer often takes prototype as working software
  - Developers often polish prototype as working software

THE PROTOTYPING PARADIGM

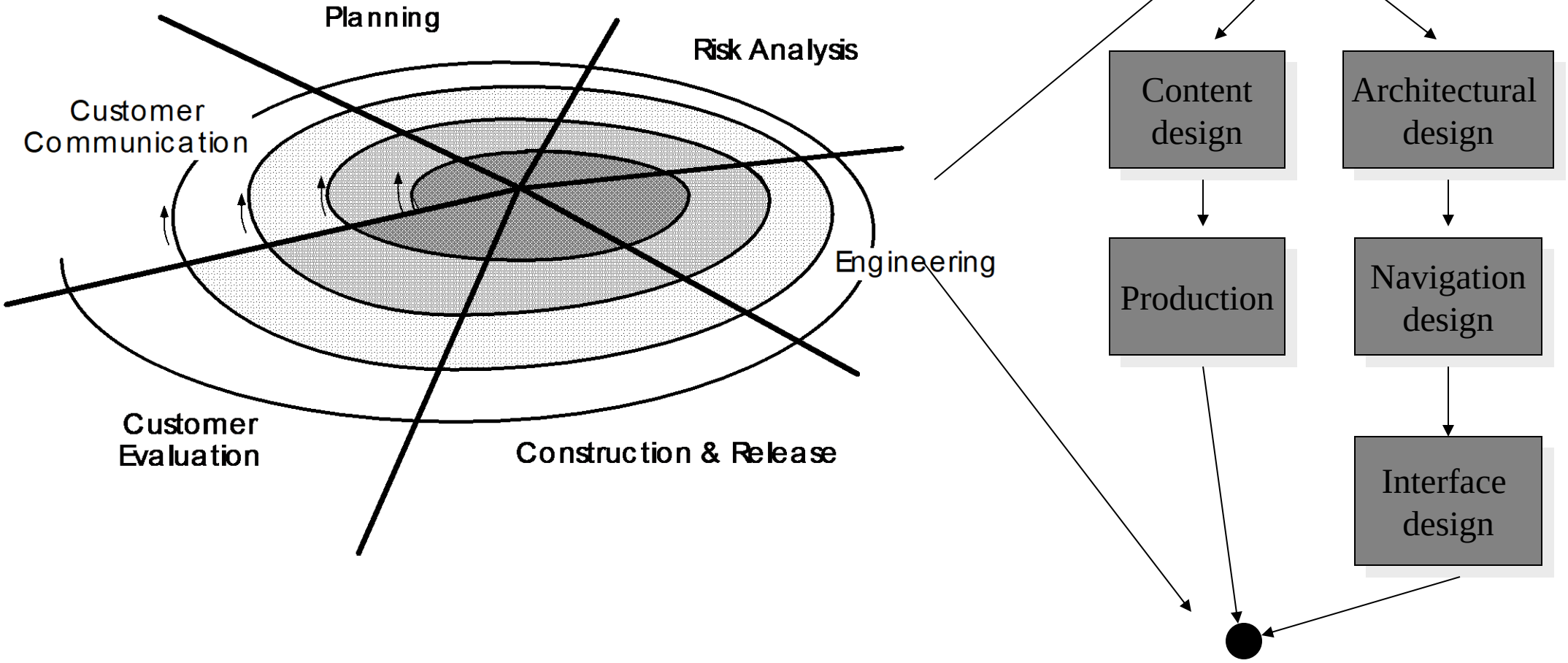


THE COMPONENT ASSEMBLY MODEL





WEB ENGINEERING PROCESS MODEL



WEEK 3

INTRODUCTION

- **Why Requirements Matter**:
  Requirements are the foundation of successful software development. They define what the system must do and serve as a reference throughout the project lifecycle.
- **Challenges in Capturing Requirements**:
  - Unclear needs from stakeholders.
  - Changing requirements.
  - Communication gaps between users and developers.

WHAT IS REQUIREMENTS ELICITATION?

- **Definition**: The process of gathering requirements from stakeholders and understanding their needs.
- **Goals**:
  - Identify functional and non-functional requirements.
  - Understand business objectives and constraints.

ELICITATION TECHNIQUES

Interviews:

- **Purpose**: Direct communication with stakeholders.
- **Best Practices**:
  - Prepare open-ended questions.
  - Record discussions for reference.

Surveys/Questionnaires:

- **Purpose**: Collect input from a large group.
- **When to Use**: Stakeholders are geographically dispersed.

Workshops:

- **Purpose**: Collaborative discussions to identify requirements.
- **Outcome**: Rapid agreement on critical features.

Observation (Shadowing):

- **Purpose**: Understand user behavior by watching them perform tasks.
- **Best For**: Systems with heavy user interaction.

Prototyping:

- **Purpose**: Create a visual mock-up to clarify requirements.
- **Outcome**: Immediate feedback from users.

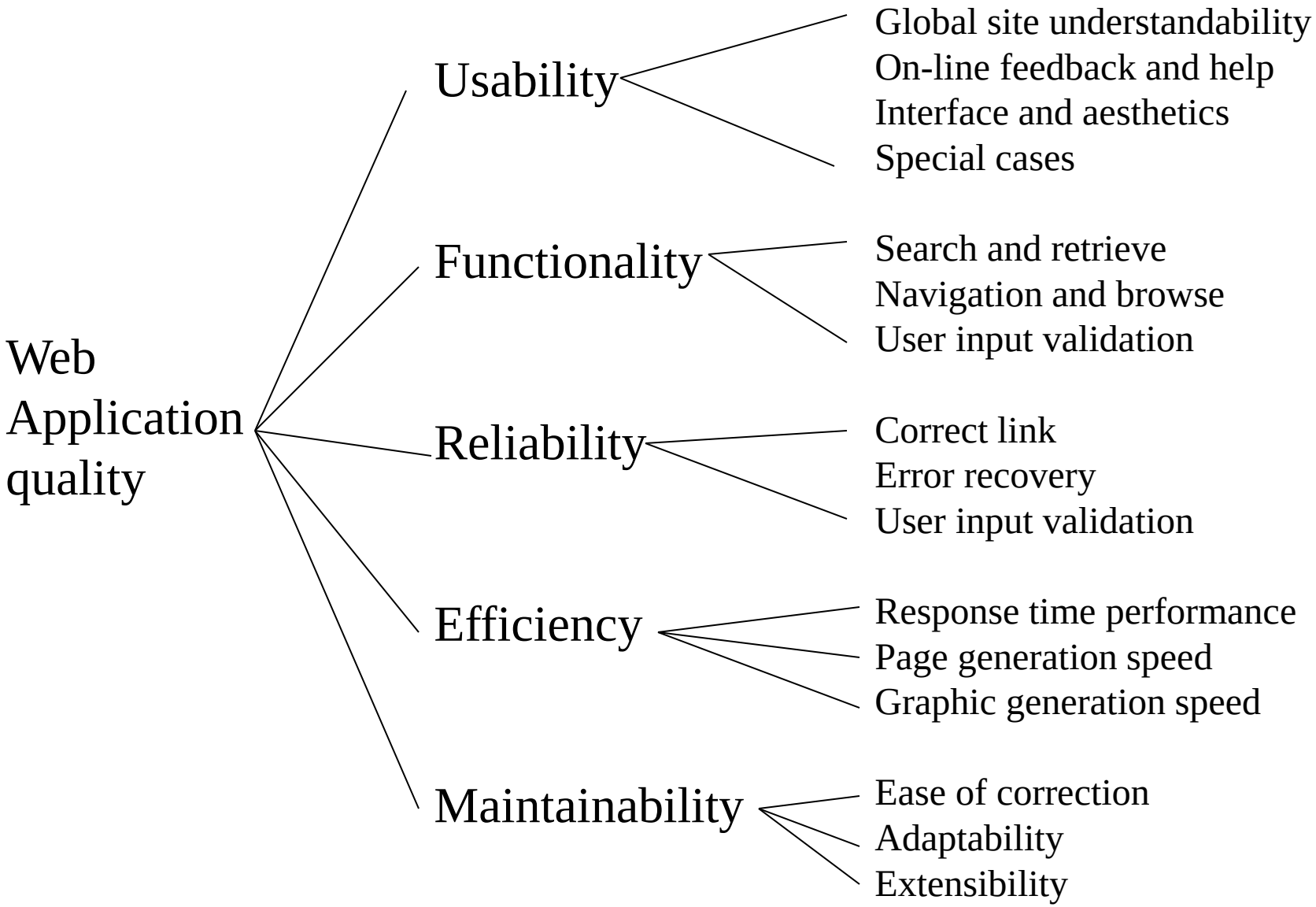
Document Analysis:

- **Purpose**: Review existing system documents, manuals, or policies.
- **When to Use**: Projects that build on or replace existing systems.

REQUIREMENTS SPECIFICATION

- **Definition**: The process of documenting and formalizing requirements into a structured format.
- **Key Objectives**:
  - Ensure requirements are **clear**, **consistent**, and **testable**.
  - Serve as a contract between stakeholders and the development team.

QUALITY REQUIREMENTS



WEEK 4

IMPORTANCE OF SYSTEM DESIGN AND ARCHITECTURE

Importance of System Design and Architecture

1. **Scalability**: Ensures the system can handle growth.
2. **Performance**: Optimizes speed and efficiency.
3. **Maintainability**: Simplifies updates and bug fixes.
4. **Reliability**: Builds robust systems that function under various conditions.

STEPS IN SYSTEM DESIGN AND ARCHITECTURE

Steps in System Design and Architecture

Understand Requirements:

- Functional and non-functional requirements.

Define System Goals:

- Scalability, reliability, performance.

Design Components:

- Identify modules, their roles, and how they interact.

Choose Technologies:

- Based on system needs (e.g., databases, frameworks, programming languages).

Create Diagrams:

- Use architectural diagrams like UML, sequence diagrams, or flowcharts to visualize the design.

Challenges in System Design and Architecture

I. Balancing trade-offs between performance and scalability.
II. Adapting to changing requirements.
III. Ensuring compatibility between components.

WEEK 5

Why Coding Standards Matter?

Coding standards ensure consistency, readability, and maintainability across software projects, especially in team settings.

Why Best Practices are Important?

They help produce high-quality, efficient, and reliable code, minimizing errors and technical debt.

WHAT ARE CODING STANDARDS?

What Are Best Practices in Coding?

Best Practices are proven methods to improve the efficiency and quality of development.

Write Clean Code

I. Follow the principle: *"Code is read more often than it is written."*
II. Remove unused variables and functions.

Follow DRY and KISS Principles

DRY (Don't Repeat Yourself): Reuse code through functions and modules.
KISS (Keep It Simple, Stupid): Avoid over-complicating solutions.

Optimize Performance

I. Avoid unnecessary loops and redundant computations.
II. Use efficient data structures (e.g., HashMaps over arrays for quick lookups).

Test Early and Often

I. Write unit tests to verify functionality.
II. Use automated testing tools like Selenium or JUnit.

Secure Your Code

I. Validate user inputs to prevent SQL injection and XSS attacks.
II. Avoid hardcoding sensitive information like passwords and keys.

TOOLS TO ENFORCE STANDARDS AND BEST PRACTICES

Linters and Formatters:

Python: Pylint, Black.

JavaScript: ESLint, Prettier.

Code Quality Checkers:

SonarQube, Codacy.

Integrated Development Environments (IDEs):

Use IntelliJ IDEA, VS Code, or Eclipse for built-in checks.

BENEFITS OF IMPLEMENTING STANDARDS AND PRACTICES

Benefits of Implementing Standards and Practices

1. **Improved Team Collaboration**: Everyone writes code in a uniform style.
2. **Easier Debugging**: Clean and consistent code is easier to understand and fix.
3. **Reduced Technical Debt**: Prevents accumulation of messy code that requires later rework.
4. **Enhanced Code Reusability**: Well-structured code can be reused in other projects.
5. **Increased Project Success Rate**: Higher-quality code leads to fewer bugs and smoother deployments.

Coding Standards

Consistent Naming Conventions:
- Use meaningful names (e.g., calculateTotal instead of ct).
- Follow established patterns like camelCase for variables and PascalCase for class names.

Proper Indentation and Formatting:
- Use consistent indentation (e.g., 2 or 4 spaces).
- Keep lines of code short (preferably <80-100 characters).

Commenting and Documentation:
- Write comments to explain **why**, not just **what**.
- Use tools like Javadoc or Docstrings for formal documentation.

Error Handling:
- Use try-catch blocks to handle exceptions gracefully.
- Always log errors for debugging purposes.

Version Control and Code Reviews:
- Use Git or other version control systems.
- Ensure peer reviews for quality assurance.

WEEK 6

TESTING OBJECTIVES

- Testing is the process of executing a program with the intent of finding errors.
- A good test case is one with a high probability of finding an as-yet undiscovered error.
- A successful test is one that discovers an as-yet-undiscovered error.

TESTING PRINCIPLES

- All tests should be traceable to customer requirements.
- Tests should be planned before testing begins.
- 80% of all errors are in 20% of the code.
- Testing should begin in the small and progress to the large.
- Exhaustive testing is not possible.
- Testing should be conducted by an independent third party if possible.

SOFTWARE TESTABILITY CHECKLIST - 1

- Operability
  - if it works better it can be tested more efficiently
- Observability
  - Source code is accessible
  - System states and variables are visible or queriable during execution
- Controllability
  - if software can be controlled better the it is more that testing can be automated and optimized

SOFTWARE TESTABILITY CHECKLIST - 2

- Decomposability
  - controlling the scope of testing allows problems to be isolated quickly and retested intelligently
- Stability
  - the fewer the changes, the fewer the disruptions to testing
- Understandability
  - the more information that is known, the smarter the testing can be done

GOOD TEST ATTRIBUTES

- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be best of breed.
- A good test should not be too simple or too complex.

TEST STRATEGIES

- Black-box or behavioral testing
  - knowing the specified function a product is to perform and demonstrating correct operation based solely on its specification without regard for its internal logic
- White-box or glass-box testing
  - knowing the internal workings of a product, tests are performed to check the workings of all possible logic paths

DETAILED TEST CASE DESIGN

- A combination of black box and white techniques are applied depending on the phase of testing - generally white box at the unit testing level, black box during acceptance testing, and both during integration and system testing.

Black Box Methods

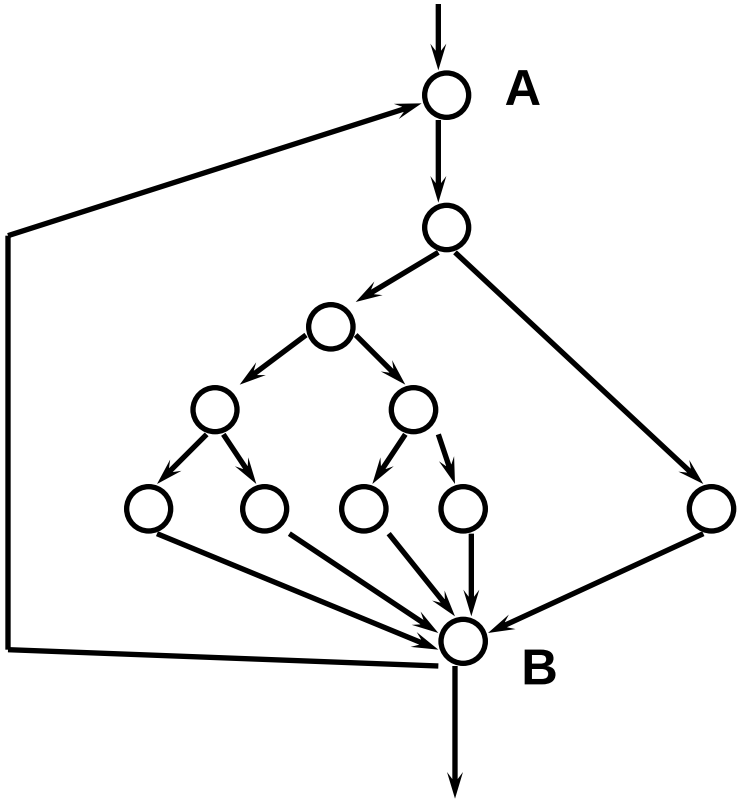
- Equivalence partitioning
- Boundary value analysis
- Cause Effect graphing
- Error Guessing

White Box Methods

- Statement Coverage
- Branch Coverage
- Condition Coverage
- Branch/Condition Coverage
- Multiple-condition Coverage

BASIS PATH TESTING

- White-box technique usually based on the program flow graph
- Determine the basis set of linearly independent paths (the cardinality of this set is the program cyclomatic complexity)
- Cyclomatic complexity is a software metric that provides a quantitative measures of the logical complexity of a program
- Prepare test cases that will force the execution of each path in the basis set.



- The number of possible paths through a program can be determined by considering the control-flow graph of the program.
- Write tests to exercise all possible paths

CONTROL STRUCTURE TESTING - 1

- White-box techniques focusing on control structures present in the software
- Condition testing (e.g. branch testing)
  - focuses on testing each decision statement in a software module
  - it is important to ensure coverage of all logical combinations of data

```
static double calc( int m, int n,
    double x) {

    if (( m<1)  &  (n==0))

      x = x /conv2real(m);

    if (( m==2)   |  (x>1.0))

      x = x+1.0;

    return x;

}
```

DATA FLOW TESTING

- selects test paths based according to the locations of variable definitions and uses in the program (e.g. definition use chains)

```
Proc x
 B1;
   do while C1
     if C2
       then
         if C4
           then B4;
           else B5;
         endif;
       else
         if C3
           then B2;
           else B3;
         endif;
       endif;
  enddo
  B6;
End proc;
```

- Simple Loops
  - Skip the loop entirely
  - Only one pass through the loop
  - Two passes through the loop
  - M passes through the loop
  - N-1,n,n+1 passes through the loop
- Nested Loops
  - Start at innermost loop
  - Work outward

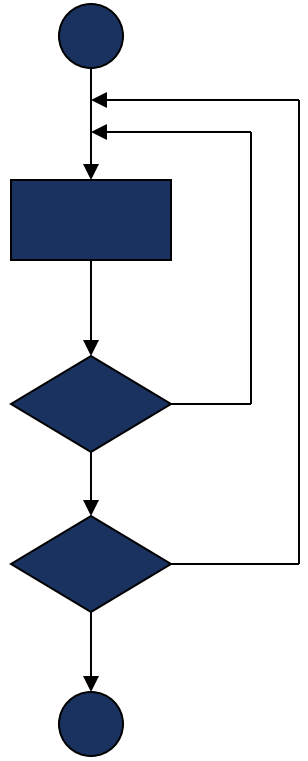


Fig: Nested loop

TEST CASES

- question: when are test cases designed?
- hint: most are designed at the same time as the user manual or guide
- answer: when the requirements analysis and spec are written not after the code is perfected
- early test cases aim to find bugs i.e. defect testing
- later ones are concerned less with defects and more with validation - meeting the users' real needs

MAJOR CONCEPTS IN TESTING

- We can consider testing from two main points of view:
  - strategies in the testing process:
    - black box (functional) testing
    - white box (structural) testing
    - top down testing
    - bottom up testing
    - stress testing
  - stages in the testing process:
    - unit testing
    - system testing
    - acceptance testing (alpha & beta testing)
    - regression testing

- Black-box testing for safety critical systems in which independently developed implementations of redundant systems are tested for conformance to specifications
- Often equivalence class partitioning is used to develop a common set of test cases for each implementation

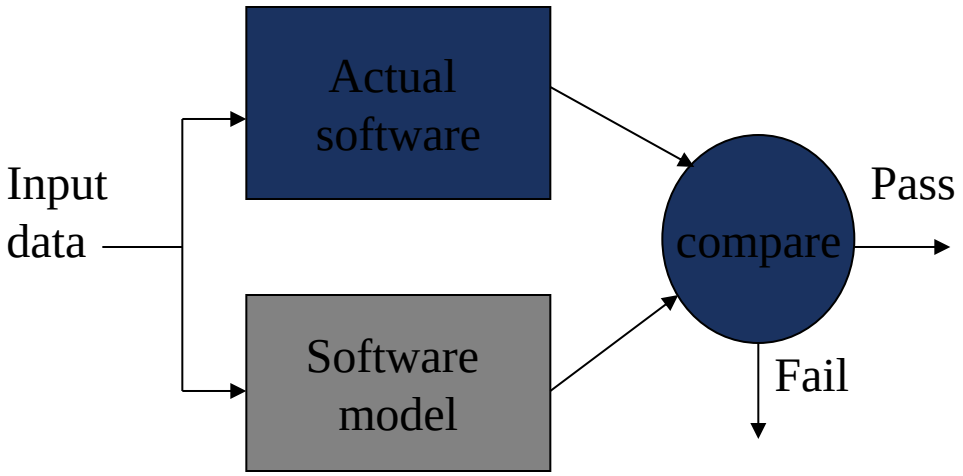


Fig: comparison testing

WEEK 7

MANAGEMENT SPECTRUM

- Effective software project management focus on four P's:
  - People
  - Product
  - Process
  - Project

PEOPLE

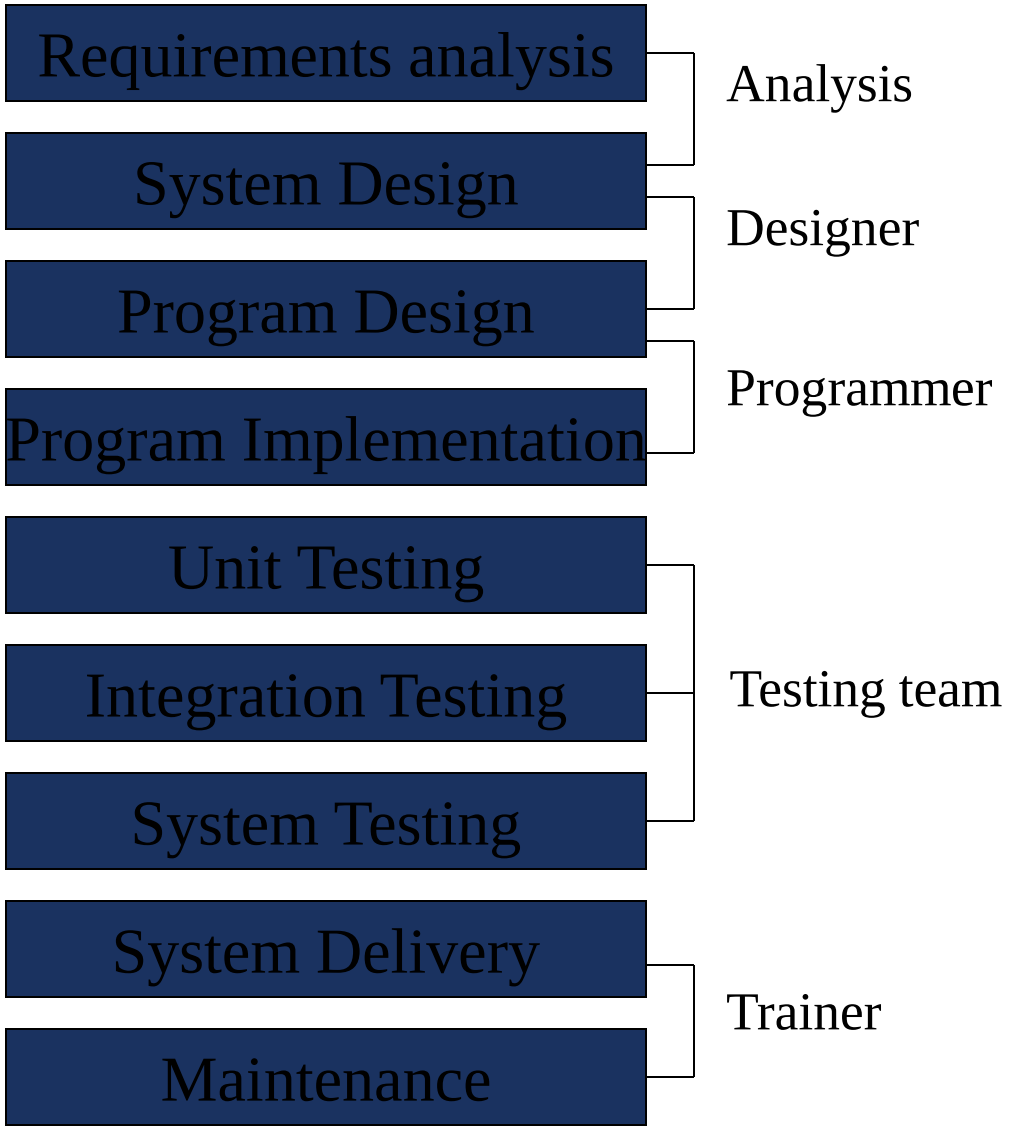
- Stakeholders
  - Senior managers - define business issues
  - Project/technical managers - plan, motivate, organize and control practitioners
  - Practitioners - deliver the technical skills i.e. developers
  - Customers - specify requirements, have interest in the outcome
  - End users - interact with the software

TEAM LEADERS

- Leadership model
  - Motivation - encourage technical people to produce their best ability
  - Organization - mould/invent existing process for fruitful outcome
  - Ideas or innovation - encourage people to create and feel creative
  - Managerial Identity - have confidence and take charge of the project
- Problem solving -
  - diagnose relative technical and organizational issues and
  - systematically structure a solution or properly motivate others to develop the solution
- Achievement - optimize productivity of a project team
- Influence team building - read people and remain control in high-stress situation

SOFTWARE TEAM ROLES

- The best team structure depends on management style
- Have trust in one another
- Distribution of skills must to appropriate to the problem
- Help create a team that exhibits cohesiveness
- Competency coupled with group collaboration is a critical success factor for the team



FACTORS AFFECTING TEAM ORGANIZATION

- Difficulty of problem to be solved
- Size of resulting program
- Team lifetime
- Degree to which problem can be modularized
- Required quality and reliability of the system to be built
- Rigidity of the delivery date
- Degree of communication required for the project

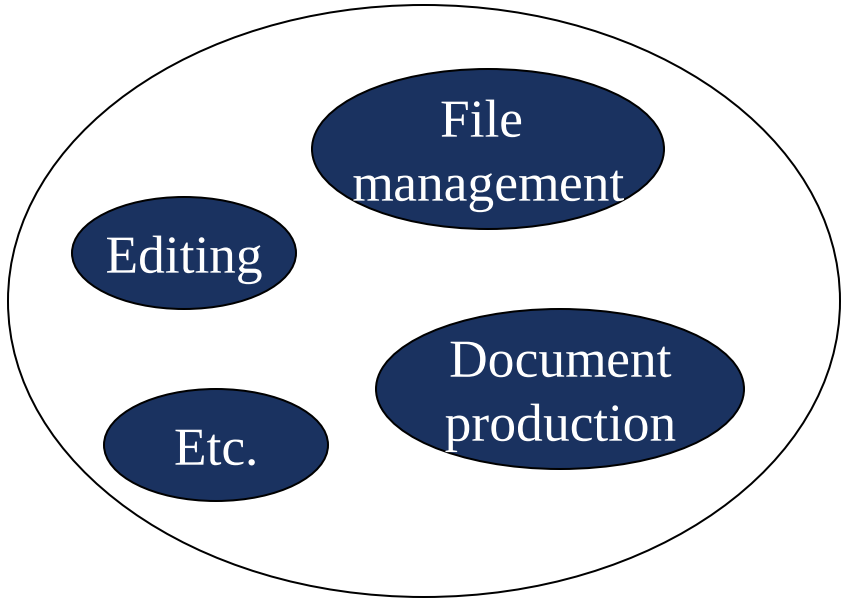
COMMUNICATION AND COORDINATION

- Formal, impersonal approaches
  - documents, milestones, memos
- Formal interpersonal approaches
  - review meetings, inspections
- Informal interpersonal approaches
  - information meetings, problem solving
- Electronic communication
  - e-mail, bulletin boards, video conferencing
- Interpersonal network
  - discussion with people outside project team

SOFTWARE SCOPE

- Context
  - How does the software fit into the larger system?
  - What constraints are imposed?
- Information objectives
  - What customer visible data objects are produced as output
  - What data objects are required for input

- Partition the complex problem into smaller manageable problems
- Focus is on functionality to be delivered and the process used to deliver it
- Example:



Word processor

PROCESS CONSIDERATIONS - 1

- Process model chosen must be appropriate for the:
  - customers
  - developers
  - characteristics of the product
  - project development environment
- Project planning begins with melding the product and the process

PROCESS CONSIDERATIONS - 2

- Each function to be engineered must pass though the set of framework activities defined for a software organization
- Work tasks may vary but the common process framework (CPF) is invariant (e.g. size does not matter)
- Software engineer's task is to estimate the resources required to move each function though the framework activities to produce each work product

CUSTOMER COMMUNICATION ACTIVITY

- Develop list of communication issues
- Meet with customer to address clarification issues
- Jointly develop a statement of scope (Statement of Work)
- Review statement of scope/work (SoW) with all concerned
- Modify Software as required

TASKS FOR CUSTOMER COMMUNICATION

- Review customer request
- Plan and schedule formal meeting with customer (regular)
- Conduct research to specify the proposed solution and existing approaches
- Prepare working doc and agenda for the formal meeting
- Conduct the meeting
- Jointly develop mini-scope
- Review SoW with all concerned
- Modify SoW doc as required

PROJECT

- Planning
- Monitoring
- Controlling

MANAGING THE PROJECT

- Start on the right foot
- Maintain momentum
- Track progress
- Make smart decisions
- Conduct a postmortem analysis

5WHH PRINCIPLE

Why?	Why is the system being developed?	This focuses a team on the business reasons for developing the software.
What?	What will be done?	This is the guiding principle in determining the tasks that need to be completed.
When?	When will it be completed?	This includes important milestones and the timeline for the project.
Who?	Who is responsible for each function?	This is where you determine which team member takes on which responsibilities. You may also identify external stakeholders with a claim in the project.
Where?	Where are they organizationally located?	This step gives you time to determine what other stakeholders have a role in the project and where they are found.
How?	How will the job be done technically and managerially?	In this step, a strategy for developing the software and managing the project is concluded upon.
How Much?	How much of each resource is needed?	The goal of this step is to figure out the amount of resources necessary to complete the project.

WEEK 8

VERSION CONTROL SYSTEMS

1. Purpose of Version Control

Tracks changes made to files over time.

Allows collaboration among multiple developers.

Facilitates recovery by rolling back to earlier versions.

2. Types of Version Control Systems

Centralized VCS A single repository stored on a server.

Changes are committed to the central server.

Distributed VCS Every developer has a complete local copy of the repository.

Changes can be made locally and later synchronized with others.

3. Key VCS Features

Commit: Save changes to the repository.

Branch: Create isolated environments to work on new features or fixes.

Merge: Integrate changes from different branches.

Revert: Roll back to a previous version of the code.

BEST PRACTICES FOR USING VERSION CONTROL SYSTEMS

1. **Commit Often**:
    1. Make small, frequent commits with clear messages.
2. **Use Branching**:
    1. Separate new features or bug fixes into branches.
3. **Merge Carefully**:
    1. Review and test changes before merging to avoid introducing bugs.
4. **Write Clear Commit Messages**:
    1. Follow a consistent format for easy tracking.
5. **Collaborate with Pull Requests**:
    1. Use tools like GitHub to request reviews and discuss changes before merging.
6. **Backup the Repository**:
    1. Always have a remote repository (e.g., GitHub, GitLab) for disaster recovery.

Managing Software Configurations

- **Why Configuration Management is Important**

•Tracks versions of not just code but also environment settings, database schemas, and build scripts.

•Ensures consistent deployment across different environments (development, testing, production).

- **Tools for Configuration Management**

Git: For source code versioning.

Ansible, Chef, Puppet: Automates configuration and deployment of environments.

Docker: Manages containerized environments for consistency across systems.

- **Configuration Best Practices**

1.Use Environment Files:

•Separate configurations (e.g., API keys, URLs) from the codebase.

2.Automate Configuration Updates:

•Use scripts or tools to ensure consistent configurations across environments.

3.Document Configuration Changes:

•Maintain a log of what changed and why, for debugging and audits.

WEEK 9

WHY DO WE MEASURE?

- To characterize
  - "How can we characterize the robustness of our system?"
- To evaluate
  - "How can we demonstrate we meet our robustness goals?"
- To predict

  "How will the software behave if the number of current transactions is doubled?"
- To improve

  "What is the gap between current and desired performance?"

SOFTWARE METRICS

Information from measurement can be used for:

- Continuous improvement of a process
- Estimation
- Quality control
- Productivity assessment

MEASUREMENTS, MEASURES, AND INDICATORS

- *measurement* = action taken
  "Count the avg. defects / 1K LOC."

- *measure* = a measurement result
  "1.2 defects / 1K LOC"

- *indicator* = comparison (trend)
  "v1.1 has .5 fewer defects / 1K LOC"

OUTCOME MEASURES

- Errors noted before delivery
- Errors noted after delivery
- Work products delivered
- Human effort expended
- Calendar time elapsed
- Schedule conformance
- Effort on “umbrella” activities

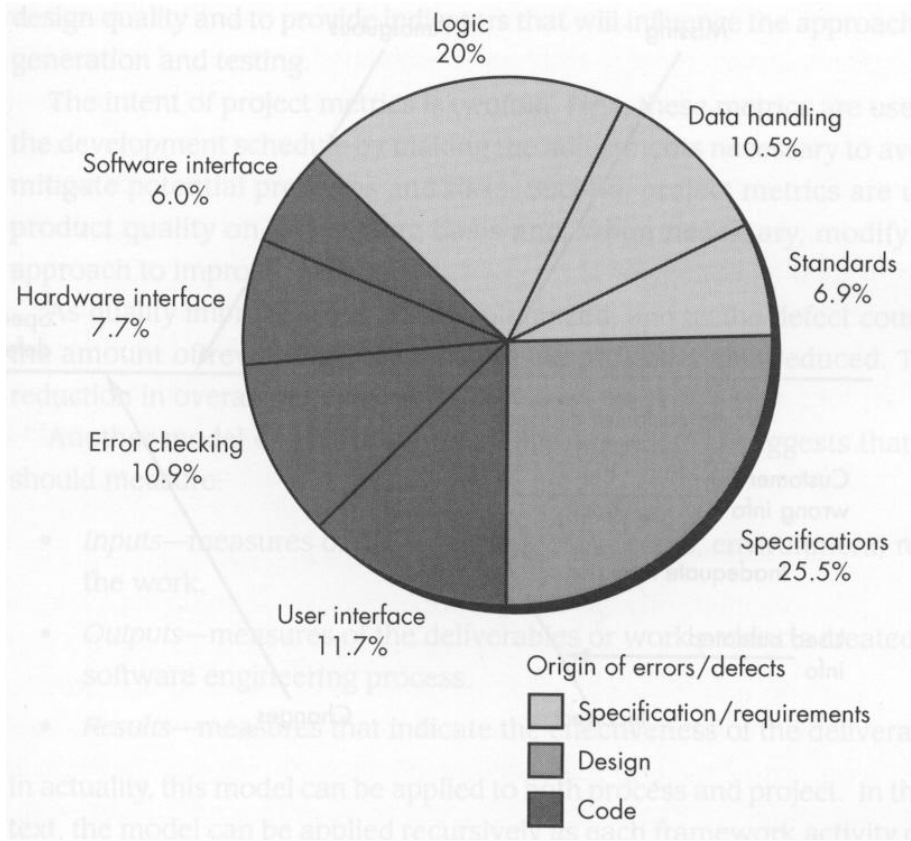
PROCESS METRICS

- Private process metrics
  - (e.g. defect rates by individual or module) are known only to the individual or team concerned.
- Public process metrics
  - enable organizations to make strategic changes to improve the software process.
- Metrics should not be used to evaluate the performance of individuals.
- Statistical software process improvement helps an organization to discover its strengths and weaknesses.

STATISTICAL SOFTWARE PROCESS IMPROVEMENT (SSPI)

- Categorize errors by origin (specification, logic, etc.)
- Estimate cost to correct
- Sort according to frequency
- Estimate cost of each error type
- Find highest-cost problems
- Prioritize debugging efforts

Defects and Their Origin



ERROR ANALYSIS

- "Fishbone" diagram used to analyze the *causes* of defects
- The analysis can be used to derive indicators for future improvements

DIRECT MEASURES

- Cost
- Effort applied
- LOC (lines of code)
- Execution speed
- Memory size
- Defects reported

INDIRECT MEASURES

- Functionality
- Quality
- Complexity
- Efficiency
- Reliability
- Maintainability

SIZE-ORIENTED METRICS

- Errors per KLOC
- Defects per KLOC
- $ per KLOC
- Documentation pages per KLOC
- Errors per person-month
- LOC per person-month
- $ per documentation page

IS LOC A GOOD MEASURE?

Lines of code are easily counted, but…

- LOC not necessarily related to quality
- Programming languages differ widely in LOC per functional requirement
- Difficult to estimate LOC
- There is more to SE than writing code!

WHAT LOC CAN'T MEASURE...

- People factors (team size, skill)
- Problem factors (complexity, change)
- Process factors (techniques, tools)
- Product factors (reliability, performance)
- Resource factors (people, tools)

METRICS FOR SOFTWARE QUALITY

- Software is only as good as the quality of...
  - the requirements description
  - the design of the solution
  - the code / program produced
  - the tests used to find errors
- QA = life-cycle task, not just a “finishing” activity
- QA must address process, too!

MEASURING QUALITY

- Correctness
  - defects per KLOC?
  - errors per inputs processed?
- Maintainability
  - mean time to change (MTTC)?

  (Note: changes to software vary widely in scope and complexity!)
- Integrity
  - assess threats, security of software

SOFTWARE QUALITY METRICS

- Factors assessing software quality come from three distinct points of view (product operation, product revision, product modification).
- Defect removal efficiency (DRE) is a measure of the filtering ability of the quality assurance and control activities as they are applied through out the process framework.

WEEK 10

What is Software Maintenance?

The process of updating and improving software after deployment to fix issues, adapt to new environments, and enhance functionality.

What is Software Evolution?

The continuous process of software adaptation and enhancement to meet changing user needs and technological advancements.

IMPORTANCE OF SOFTWARE MAINTENANCE

1. Software is rarely "complete" after its initial release.
2. Maintenance ensures:

Reliability: Fixes bugs to maintain stability.

Relevance: Adapts software to new technologies and user requirements.

Security: Resolves vulnerabilities and prevents exploitation.

TASK OF SOFTWARE MAINTENANCE

Corrective Maintenance:

1. Fixes bugs and defects reported by users or found during operation.

Adaptive Maintenance:

1. Updates software to remain compatible with new hardware, operating systems, or regulations.
2. Example: Updating an app to support the latest iOS version.

Perfective Maintenance:

1. Enhances or improves software functionality based on user feedback.

Preventive Maintenance:

1. Anticipates and addresses potential future issues to improve software reliability.

MAINTENANCE PROCESSES

- **Identification and Analysis**:
  - Gather and analyze requests for changes or fixes.
- **Impact Assessment**:
  - Evaluate how the change affects the existing system.
  - Identify dependencies and risks.
- **Implementation**:
  - Develop and test the necessary changes.
  - Follow coding standards and best practices.
- **Verification and Testing**:
  - Ensure the software functions correctly after changes.
  - Test for unintended side effects.
- **Release and Deployment**:
  - Deliver the updated version to users.
- **Monitoring**:
  - Track the performance and stability of the updated software.

CHALLENGES IN SOFTWARE MAINTENANCE

1. **Increasing Complexity**:
   - ✓ Over time, software systems become harder to understand and modify.
2. **Lack of Documentation**:
   - ✓ Poor or outdated documentation makes it difficult to implement changes.
3. **Regression Issues**:
   - ✓ Fixing one issue may introduce new bugs.
4. **Resource Constraints**:
   - ✓ Limited budget or time for maintenance tasks.
5. **Understanding Legacy Code**:
   - ✓ Old or poorly written code can be hard to modify.
6. **Changing User Requirements**:
   - ✓ Balancing current fixes with evolving user demands.

EFFECTIVE WAYS OF MAINTENANCE

Write Maintainable Code:

Use clean coding practices and follow coding standards.

Maintain Proper Documentation:

Update documentation regularly to reflect changes.

Automate Testing and Deployment:

Use tools like Jenkins or GitLab CI/CD to streamline processes.

Use Version Control:

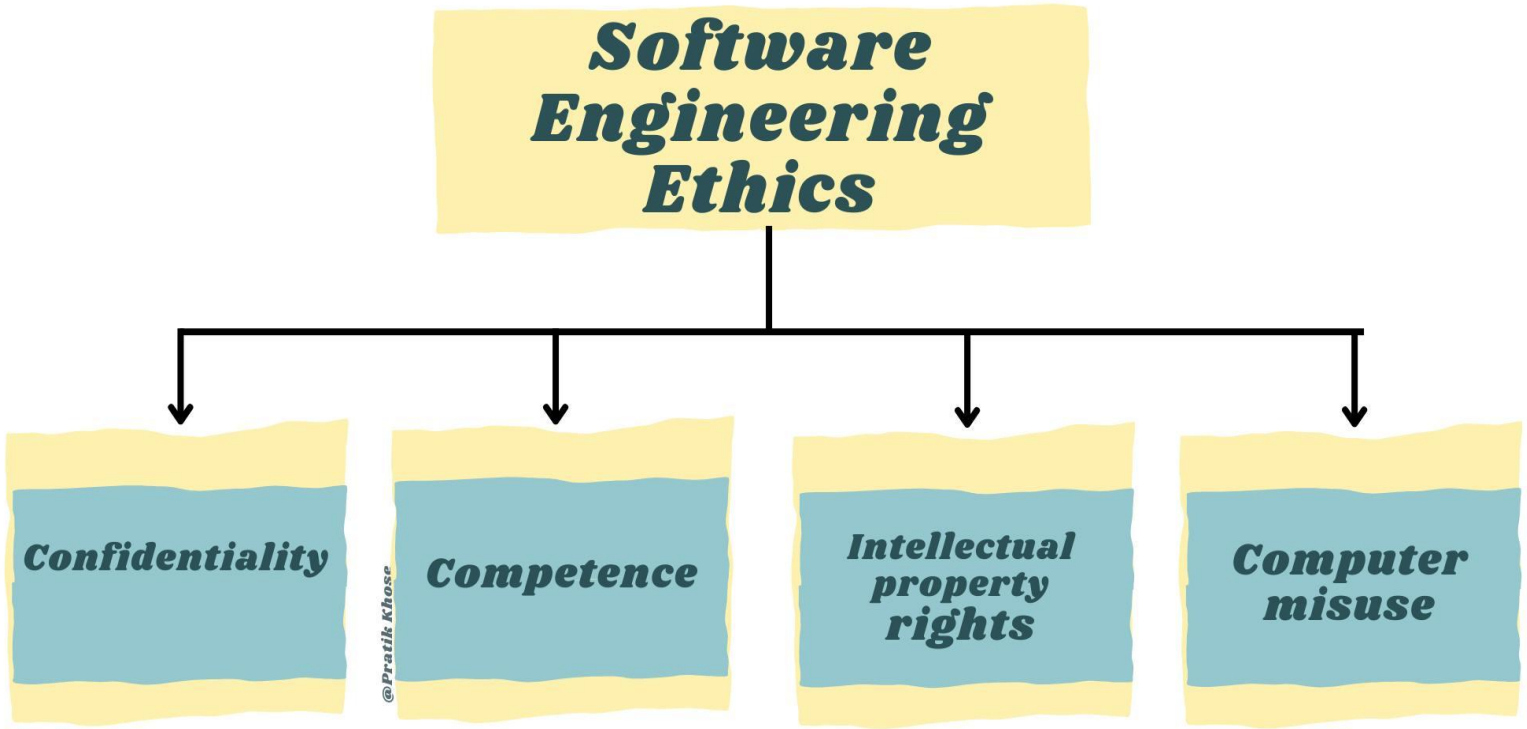
Track changes and roll back to previous versions when needed.

Regular Code Reviews:

Identify potential issues and ensure consistency.

WEEK 11

What is Software Engineering Ethics and why Engineering Ethics is important?



MAJOR ETHICAL ISSUES IN SOFTWARE ENGINEERING

- **Privacy**

Issue: Protecting users' personal information and ensuring data security.

Example: Misusing personal data or allowing unauthorized access to sensitive information.

Responsibility: Engineers must design software to safeguard user privacy and comply with data protection laws.

- **Security**

Issue: Ensuring software is secure from attacks and vulnerabilities.

Example: Failing to fix known security issues, leading to potential data breaches.

Responsibility: Software engineers must prioritize security during development, identify vulnerabilities, and apply updates promptly.

- **Intellectual Property**

Issue: Respecting others' intellectual property (IP) rights and ensuring that no plagiarism occurs.

Example: Using copyrighted code without permission or credit.

Responsibility: Engineers should give credit to authors of third-party code and follow licensing agreements.

- **Software Quality**

Issue: Creating software that is reliable, bug-free, and fit for use.

Example: Releasing software with known defects that can cause harm to users.

Responsibility: Engineers must thoroughly test and ensure software quality before release to avoid errors that can affect users.

RESPONSIBILITIES OF SOFTWARE ENGINEERS

Honesty and Integrity

Responsibility: Software engineers should be truthful about their skills, progress, and the capabilities of the software they are developing.

Accountability

Responsibility: Engineers must take responsibility for their actions, decisions, and the impact of their work.

Respect for Users and Society

Responsibility: Software engineers should consider how their software affects users and society, avoiding harm and ensuring accessibility.

Continuous Learning and Professional Development

Responsibility: Engineers should stay updated with new technologies, tools, and ethical standards.

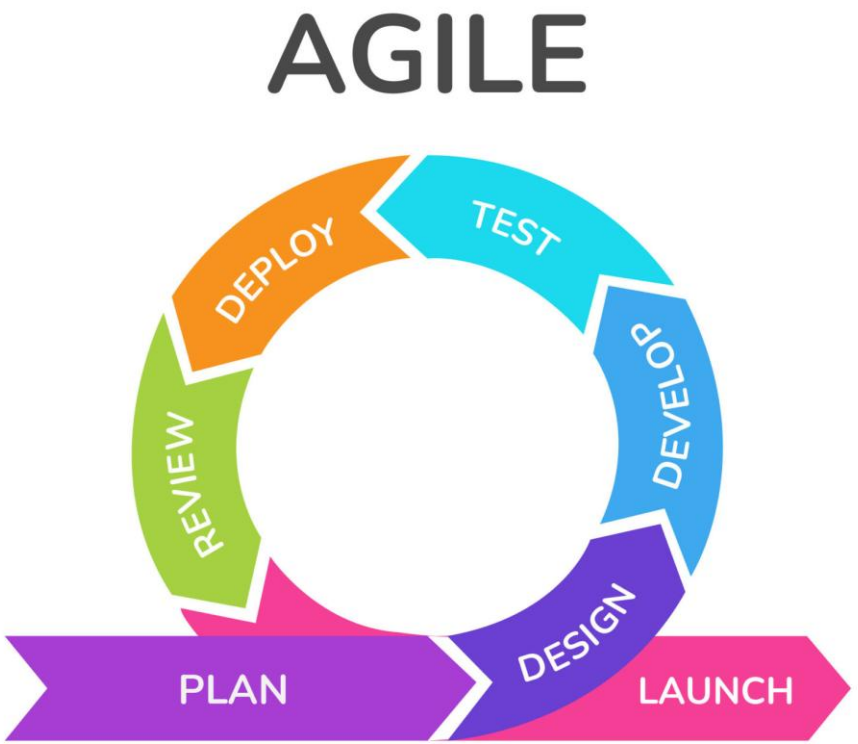
CODE OF ETHICS FOR SOFTWARE ENGINEERS

- Many professional organizations, like the **ACM** (Association for Computing Machinery) and **IEEE**, have established ethical codes that software engineers should follow.
- These codes emphasize principles like public welfare, professional competence, honesty, and integrity.

WEEK 12

AGILE DEVELOPMENT

What is Agile Development?



KEY PRINCIPLES OF AGILE

Agile is based on the **Agile Manifesto**, which emphasizes:

I. **Individuals and Interactions** over processes and tools.

II. **Working Software** over comprehensive documentation.

III. **Customer Collaboration** over contract negotiation.

IV. **Responding to Change** over following a plan.

AGILE VS WATERFALL

Agile	Waterfall
It’s iterative and it welcomes feedback at different stages	Follows a linear approach
Definition happens in chunks at the beginning of each sprint	Users need to define and delimit the complete project before they begin
Work happens in simultaneous increments	Work is sequential
Testing and validation happens after each iteration delivery	Testing happens at the end of the project
Admits and fosters changes at any stage	Changes are discouraged after the definition stage

AGILE METHODOLOGIES

1. **Scrum**
    i. Focuses on delivering software in **sprints** (short, time-boxed development cycles, usually 2-4 weeks).
    ii. Key roles: **Scrum Master** (facilitator), **Product Owner** (represents the customer), and **Development Team**.
    iii. Scrum uses ceremonies like **Daily Stand-ups**, **Sprint Planning**, **Sprint Review**, and **Sprint Retrospectives**.
2. **Kanban**
    i. Focuses on visualizing the workflow and continuously improving efficiency.
    ii. Uses a **Kanban board** to track the status of tasks (To Do, In Progress, Done).
    iii. Ideal for teams with changing priorities and continuous work.
3. **Extreme Programming (XP)**
    i. Focuses on software quality and developer collaboration.
    ii. Practices include **pair programming**, **test-driven development (TDD)**, and frequent releases.

BENEFITS AND CHALLENGES OF AGILE DEVELOPMENT

AGILE	
Benefits	**Challenges**
• Rapid and continuous delivery of working software • Amendable to changing requirements (even late in development) • Faster development cycles • Consistent feedback and collaboration • Continuous improvement/sprints • Enhanced collaboration and quality • Increased transparency • Increased project success and efficiency	• Less predictability and planning • Large cultural and organizational impacts • More difficult to transfer to new team members (owing to less documentation) • Difficult to quantify total time, resources, and effort • Selecting the most appropriate Agile methodology • Greater demands on clients

WEEK 13

SOFTWARE SECURITY

- Most security issues in software systems result from flaws in the code or design of software system.

- CERT reported over 5000 software vulnerabilities in 2005.
  - These flaws are the result of inadequate consideration of security during requirements analysis, design, implementation, and testing of software systems.

- Production of trustworthy software has become an issue.
  - The need for trustworthy systems result from the growth in complexity and connectivity of modern software systems.

- Trustworthy software relies on the education of computer scientists and software engineers.
- Most threats to our software are with anti-virus software on the host, with firewalls and intrusions prevention systems.
- Agile software development has been used by industry to create a more flexible and lean software development process.

WHAT IS SOFTWARE SECURITY

- Focus is on how security vulnerabilities arise from poor software engineering practices.
- Both coding bugs and architectural flaws introduced as sources of software vulnerabilities

Expected outcome:

- Clear understanding of the need for software security and how software security is different from security features like access control or cryptography.

THREATS AND VULNERABILITIES

- Case studies of software security exploits were studied.
- By this, the nature of threats and vulnerabilities could be understood.
- The case studies were selected mostly from web applications.
- Attack patterns were studied, which describe general methods attackers use to exploit software vulnerabilities.

Expected outcome:

- Understand common threats to web applications and common vulnerabilities written by developers.

RISK MANAGEMENT

- Security needs to be understood in terms of risk management.
- It is impossible to eliminate all risks, risks evolve with time.
- Data flow diagrams are constructed to describe architecture of application and to document trust levels and system boundaries
- These diagrams show potential entry points into the application

Expected outcome:

- Create a risk model of a web application, ranking and detailing the risks to the system’s assets.

SECURITY REQUIREMENTS

- Security requirements are based on the risk model for the application.
- Security requirements describe what software should not do.
- Attackers frequently do what the application designers assume that users never do in order to compromise an application.
- In order to understand their assumptions in designing a system, system architects need to think like an attacker.

Expected outcome:

- Construct, document, and analyze security requirements with abuse cases and constraints.

AN AGILE SECURITY PROCESS

- Combining the most compatible and beneficial activities, shown in the picture, with an Agile process.
- This create an Agile security process that implements the most cost effective benefits from the three SE process.

Product Owner	
Requirement	
Security Requirements	Role Matrix

SP Development Team	
Design	Implementation
Countermeasure graphs	Static Code Analyses
Assumption Documentation	Coding Rules
Abuse Cases	
Requirement Inspection	
Release	
Repository Improvement (retrospect meeting)	

LSV Test Team
Testing
Dynamic analyses

AN AGILE SECURITY PROCESS

The three security engineering processes are:

- Cigatel's Touchpoints
- Microsoft SDL
- Common Criteria

- To better integrate the SE activities, move them from their recommended phase and place them with the development team.
- The activity can be performed during a work package.
- For example, Abuse Cases are moved to the design phase and the coding developers should write them instead of the requirement engineers.

AN AGILE SECURITY PROCESS

- In the release phase, Repository Improvement fits very well with the retrospective meeting that development team performs at the end of a work sprint.
- Integrating these two activities is made easier by moving them to a more developer heavy phase.
- During requirement analysis, the security enhanced Agile process primary focus is:
  - To write specific requirements for security goals.
  - These will aid both programmers and testers to know what goals are required and need verification.
- Having better security specified user stories makes it easier to write, design safe software.
- The main focus for the improvements are with the developers teams and their work sprints.

KEY AGILE FRAMEWORKS FOR SOFTWARE DEVELOPMENT

Scrum

Scrum is an Agile framework that breaks down the software development process into **sprints**—time-boxed periods (usually 2-4 weeks) where specific features or tasks are completed.

Kanban

Kanban focuses on continuous delivery by visualizing the work process using a **Kanban board**, where tasks flow through columns.

Extreme Programming (XP)

XP focuses on high-quality code and customer satisfaction by practicing **frequent releases**, **pair programming**, and **test-driven development (TDD)**.

WEEK 14

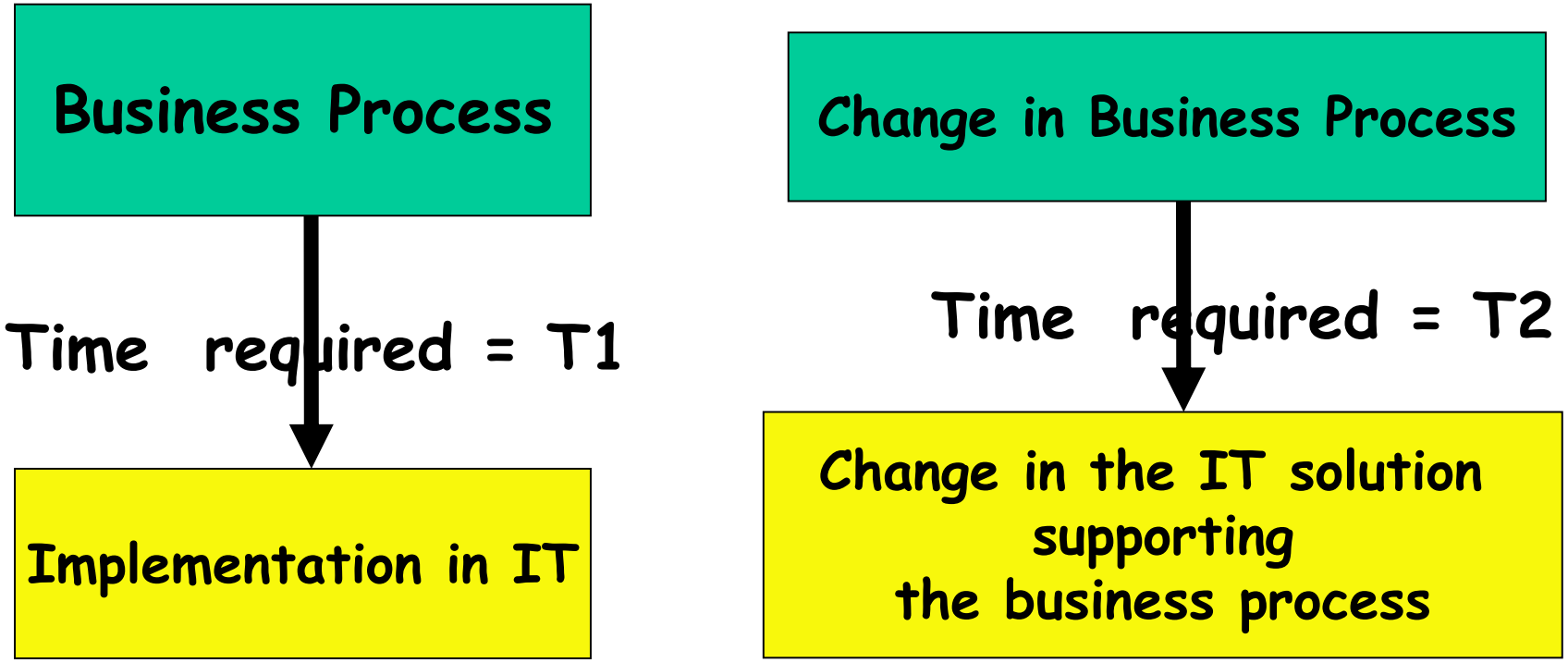
EMERGING TRENDS IN SOFTWARE ENGINEERING

- Technology developments occur:
  - To adapt to new environments
  - To respond to new challenges
- Few important developments that have occurred over the last decade or so:
  - Desktops have become more powerful and at the same time more affordable.
  - Internet has become widely accepted.
  - Mobile computing.
  - Outsourcing has become prevalent.

- Software market has two parts:
  - Products (General purpose software)
  - Services (custom software)
- Total business – appx. $1 Trillion
  - Half in products and half services
  - Services segment is growing fast

- SE goal is to develop software to satisfy user needs.
  - Either generic or one-off.
- Customer needs are considered sacrosanct and fixed:
  - Vendor has to find the solution.

BACKGROUND: A CONSTRAINT ON SOFTWARE ENGINEERING PROCESS



DESIRED GOALS

- A. Reduce T1 (and improve quality)
- B. Reduce T2
- A requires:
  - Process should efficiently capture requirements of the business and then efficiently convert it to code.
- B requires:
  - Implementation should be such that business process changes can be easily accommodated in the software solution.

CHALLENGES BEING FACED

- Delivery time requirements are shortening:
  - High business velocity requires this.
  - Software is becoming a bottleneck in implementing business process changes.
- Businesses are getting tired of software cost, late deliveries, poor quality, risk,…
  - Hardware and software cost differentials are becoming more and more glaring.

CHALLENGES BEING FACED

- Software sizes are further increasing.
- How software is to be effectively developed in following contexts is not clear:
  - Distributed platforms
  - Working with Internet
  - Distributed development teams

NOTICEABLE SOFTWARE ENGINEERING TECHNOLOGY TRENDS

- Following software engineering trends are easily noticeable:
  - Client-server (or Component-based) development
  - Service-Oriented Architecture (SOA)
  - Software as a Service (SaaS)

ARTIFICIAL INTELLIGENCE (AI) AND MACHINE LEARNING (ML) IN SOFTWARE DEVELOPMENT

Artificial Intelligence (AI): AI refers to the simulation of human intelligence in machines that are programmed to think and learn like humans.

Machine Learning (ML): ML is a subset of AI that involves training algorithms to recognize patterns in data and make predictions or decisions based on that data.

AI/ML in Software Development

- **Code Generation and Automation**:

  AI-powered tools can automatically generate code, reducing manual effort and speeding up development.

- **Bug Detection and Code Review**:

  Machine learning models can analyze code and identify potential bugs or security vulnerabilities.

- **Predictive Analytics**:

  AI can predict project timelines, risks, and outcomes by analyzing historical data from past projects.

- **Intelligent Testing**:

  AI and ML can automate testing processes by generating test cases, predicting which parts of the system need testing, and even simulating user behavior.

WEEK 15

OBJECT-ORIENTED ANALYSIS (OOA) TASKS

- Basic user requirements must be communicated between the customer and the software engineer
- Classes must be identified (e.g. define attributes and methods)
- Specify class hierarchy
- Represent object-to-object relationships
- Model object behavior
- Reapply 1 through 5 iteratively until model is complete

OOA GENERIC STEPS

- Elicit (draw out) customer requirements for system
- Identify scenarios or use cases
- Select classes and objects using basic requirements as a guide
- Identify attributes and operations for each system object
- Define structures and hierarchies that organize classes
- Build object-relationship model
- Build object-behavior model
- Review OOA model against use-cases (scenarios)

GENERIC OOA MODEL - 1

- Static view of semantic classes
  - classes based on semantics of customer requirements
- Static view of attributes
  - attributes describe classes
  - suggest operations relevant to classes
- Static view of relationships
  - represent relationships in a way that allows identification of operations and the design of a messaging approach

GENERIC OOA MODEL - 2

- Static view of behaviors
  - behaviors accommodating system usage scenarios implemented by sequences of operations
- Dynamic view of communication
  - among objects
  - based on events that cause state transitions
- Dynamic view of control and time
  - describe the nature and timing of events causing state transitions

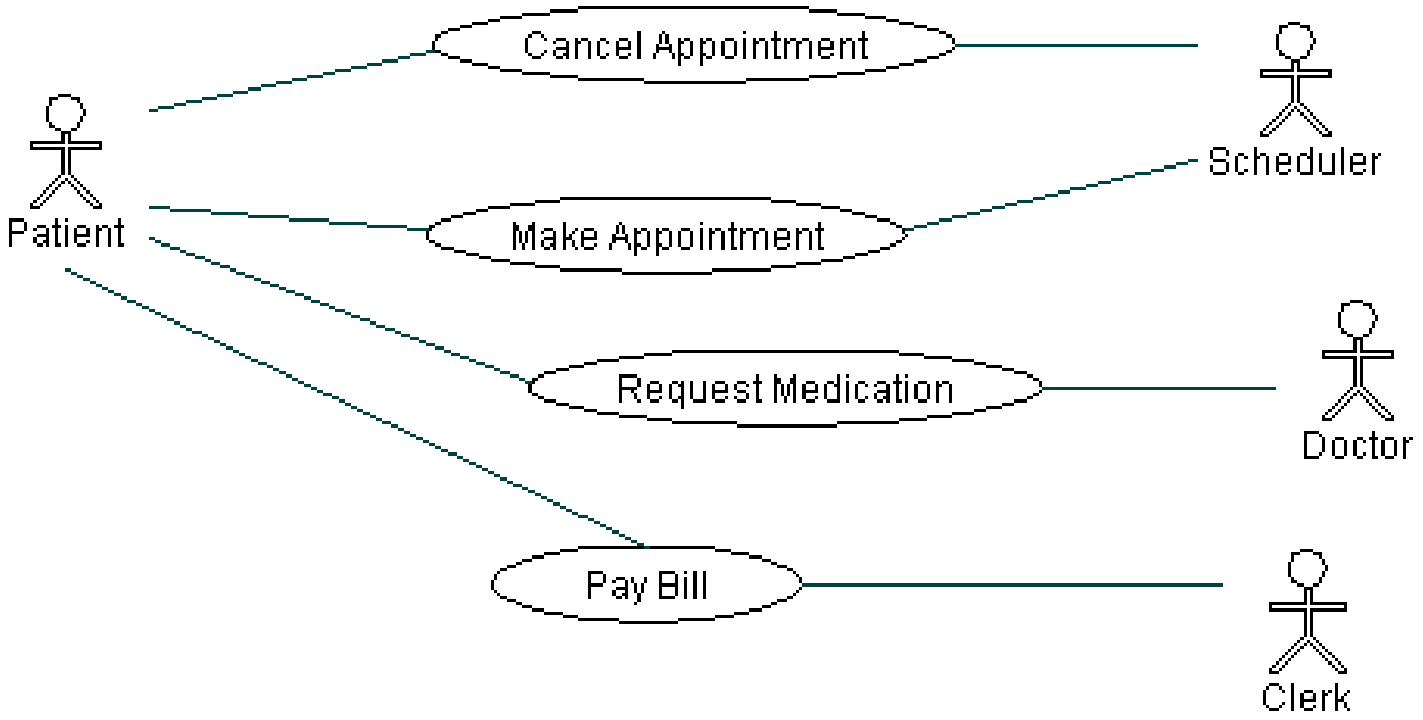
UNIFIED MODELING LANGUAGE (UML) ELEMENTS - I

- In UML, a system is represented using five different views
- User model view
  - describes usage scenarios from the end-user's perspective
- Structural model view
  - static structure of data and functionality is modeled (classes, objects, relationships)
- Behavioral model view
  - represents dynamic system aspects
  - interactions or collaborations between structural elements in the user and structural models

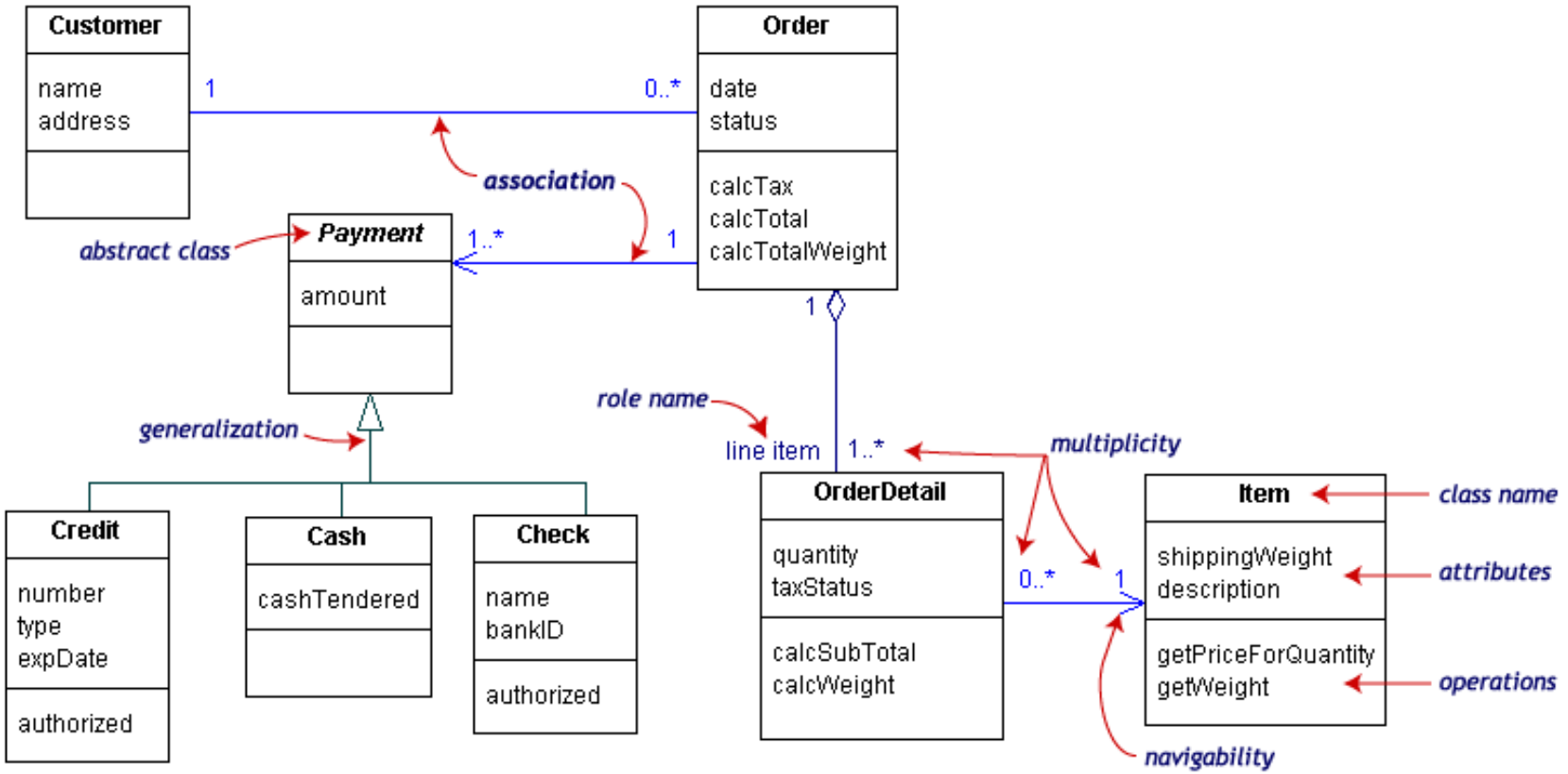
UNIFIED MODELING LANGUAGE (UML) ELEMENTS - 2

- Implementation model view
  - representing the structural and behavioral aspects of the system as they are to be built
- Environment model view
  - representation of the structural and behavioral aspects of the environment in which the system will be implemented

UML USE CASE (TOGETHERSOFT)



UML CLASS MODEL (TOGETHERSOFT)



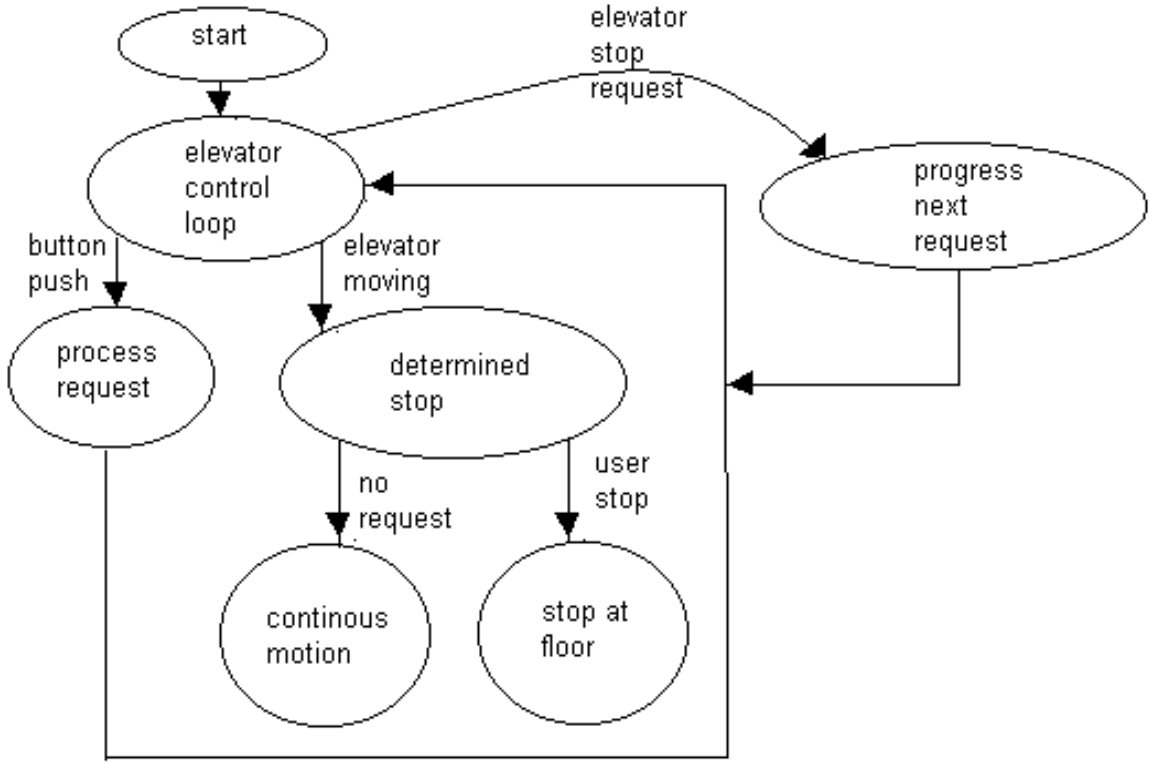
SIMPLIFIED OOA

- Class or Object Modeling
  - build an object model similar to an ER diagram
- Dynamic or Behavior Modeling
  - build a finite state machine type model
- Functional Modeling
  - similar to data flow diagram

OOA BEHAVIORAL MODEL CONSTRUCTION

- Evaluate all use-cases to understand the sequence of interaction within the system
- Identify events that drive the interaction sequence and how events relate to specific objects
- Create an event-trace for each use-case
- Build a state transition diagram for the system
- Review the object behavior model to verify accuracy and consistency

BEHAVIORAL MODEL



WEEK 16

SOFTWARE QUALITY ASSURANCE

QUALITY MANAGEMENT

- Quality management encompass
  - A software quality assurance (SQA) process
  - Specify quality assurance and quality control tasks (formal technical review and multi-tiered testing strategy)
  - Effective software engineering practices (methods and tools)
  - Control of all software work products and changes made to them

QUALITY CONCEPTS

- Quality of design
  - refers to characteristics designers specify for the end product to be constructed
- Quality of conformance
  - degree to which design specifications are followed in manufacturing the product
- Quality control
  - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications
- Quality assurance
  - auditing and reporting procedures used to provide management with data needed to make proactive decisions

QUALITY COSTS

- Prevention costs
  - quality planning, formal technical reviews, test equipment, training
- Appraisal costs - activities to gain insight into product condition
  - in-process and inter-process inspection, equipment calibration and maintenance, testing
- Failure costs
  - rework, repair, failure mode analysis
- External failure costs
  - complaint resolution, product return and replacement, help line support, warranty work

STATISTICAL QUALITY ASSURANCE

- Information about software defects is collected and categorized
- Each defect is traced back to its cause
- Using the Pareto principle (80% of the defects can be traced to 20% of the causes) isolate the "vital few" defect causes
- Move to correct the problems that caused the defects

SOFTWARE RELIABILITY

- Defined as the probability of failure free operation of a computer program in a specified environment for a specified time period
- Can be measured directly and estimated using historical and developmental data (unlike many other software quality factors)
- Software reliability problems can usually be traced back to errors in design or implementation.

SQA PLAN – 1

- Management section
  - describes the place of SQA in the structure of the organization
- Documentation section
  - describes each work product produced as part of the software process
- Standards, practices, and conventions section
  - lists all applicable standards/practices applied during the software process and any metrics to be collected as part of the software engineering work

SQA PLAN - 2

- Reviews and audits section
  - provides an overview of the approach used in the reviews and audits to be conducted during the project
- Test section
  - references the test plan and procedure document and defines test record keeping requirements
- Problem reporting and corrective action section
  - defines procedures for reporting, tracking, and resolving errors or defects, identifies organizational responsibilities for these activities
- Other
  - tools, SQA methods, change control, record keeping, training, and risk management

SIX SIGMA FOR SOFTWARE ENGINEERING

- 6σ - most widely used statistical quality assurance in industry
- Popularized by Motorola in 1980s
- Six sigma is derived from six standard deviation - 3.4 instances (defects) per million occurrences
- It defines 3 core concepts

SIX SIGMA CORE STEPS

- *Define* customer requirements, deliverables, and project goals via well defined methods of customer communication
- *Measure* the existing process and its output to determine current quality performance (collect defect metrics)
- *Analyze* defect metrics and determine the vital few causes

SIX SIGMA ADDITIONAL STEPS

- If an existing software process is in place but requires improvements:
- *Improve* the process by eliminating the root causes of defects
- *Control* the process to ensure that future work does not reintroduce the causes of defects
- It is also known as DMAIC (*define, measure, analyze, improve and control*)

- If the organisation is developing a process use:
- *Design* the process to (1) to avoid the root cause of defects and (2) to meet customer requirements
- *Verify* that the process model will, in fact, avoid defects and meet customer requirements
- This variation is known as DMADV (*define, measure, analyze, design and verify*)

ISO 9000 QUALITY STANDARDS

- ISO 9000 describes generic quality standards applicable to any business
- ISO 9001:2000 is the quality standard that applies to software engineering
- ISO 9001:2000 20 requirements that must be present for an effective quality assurance system
- ISO 9000-3 have been developed to help interpret the standard for use in the software process
- Need this certification of quality assurance to work in certain countries

WEEK 17

FINAL PROJECT PRESENTATIONS

- Students present their projects demonstrating the application of course concepts.
- Peer and instructor feedback sessions.

THANK YOU