Software Engineering Practices

Agenda

Introduction

Module1

Objectives

At the end of this module, you will be able to:

- Define what is meant by software engineering
- Distinguish types of software
- Identify the characteristics of "good software"
- Discuss on attributes which are characteristic of software products and which measures need to be undertaken to assure quality
- Identify Software product characteristics

Duration: 2 hrs

What is software engineering

Software Engineering is concerned with building software systems which are large than would normally be tackled by a single individual, uses engineering principles in the development of these systems and is made up of both technical and non-technical aspects.

-- *Sommerville*

The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them.

-- Boehm

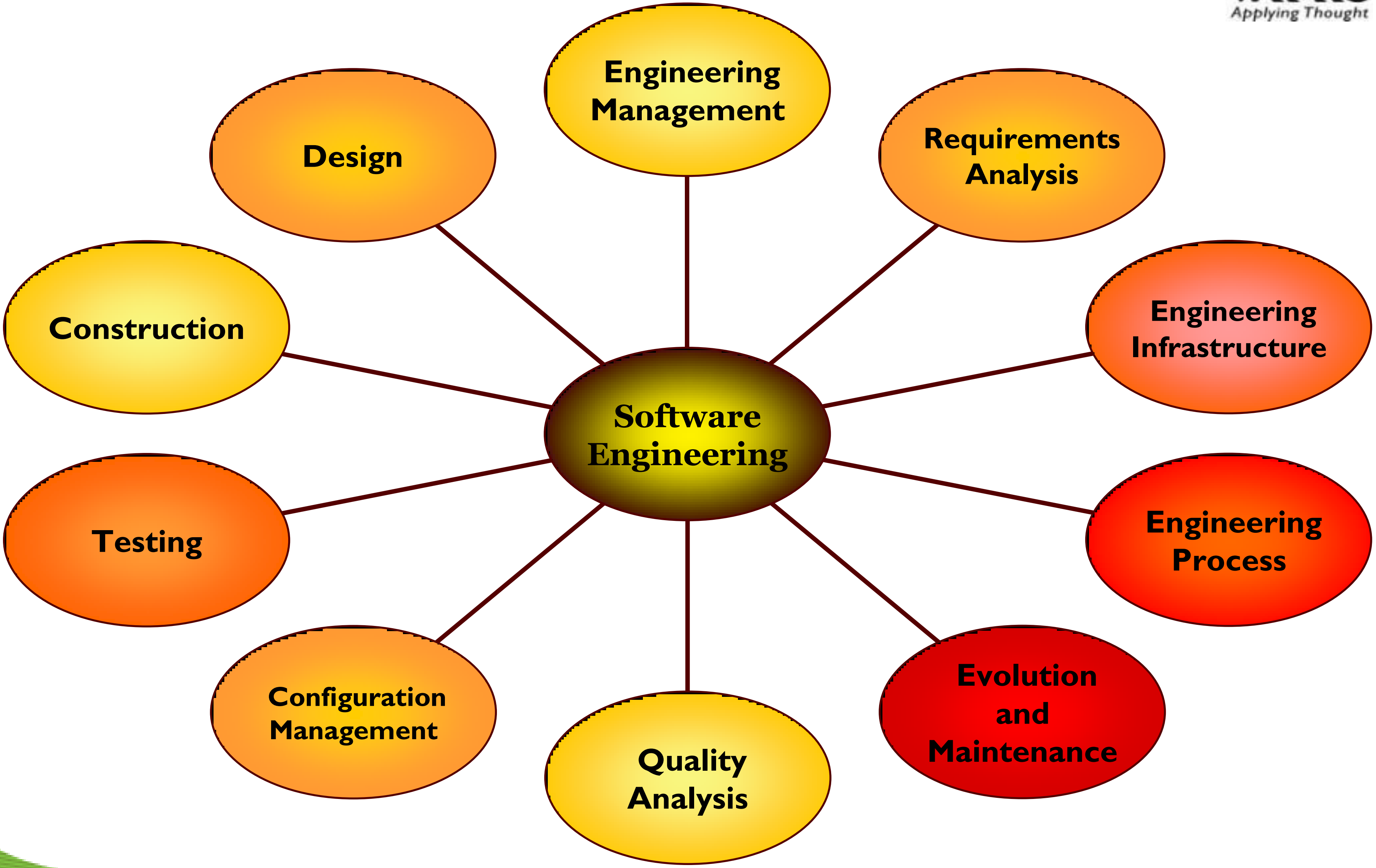
Software engineering is the practical application of scientific knowledge for the economical production and use of high-quality software

- Pomberger and Blaschek

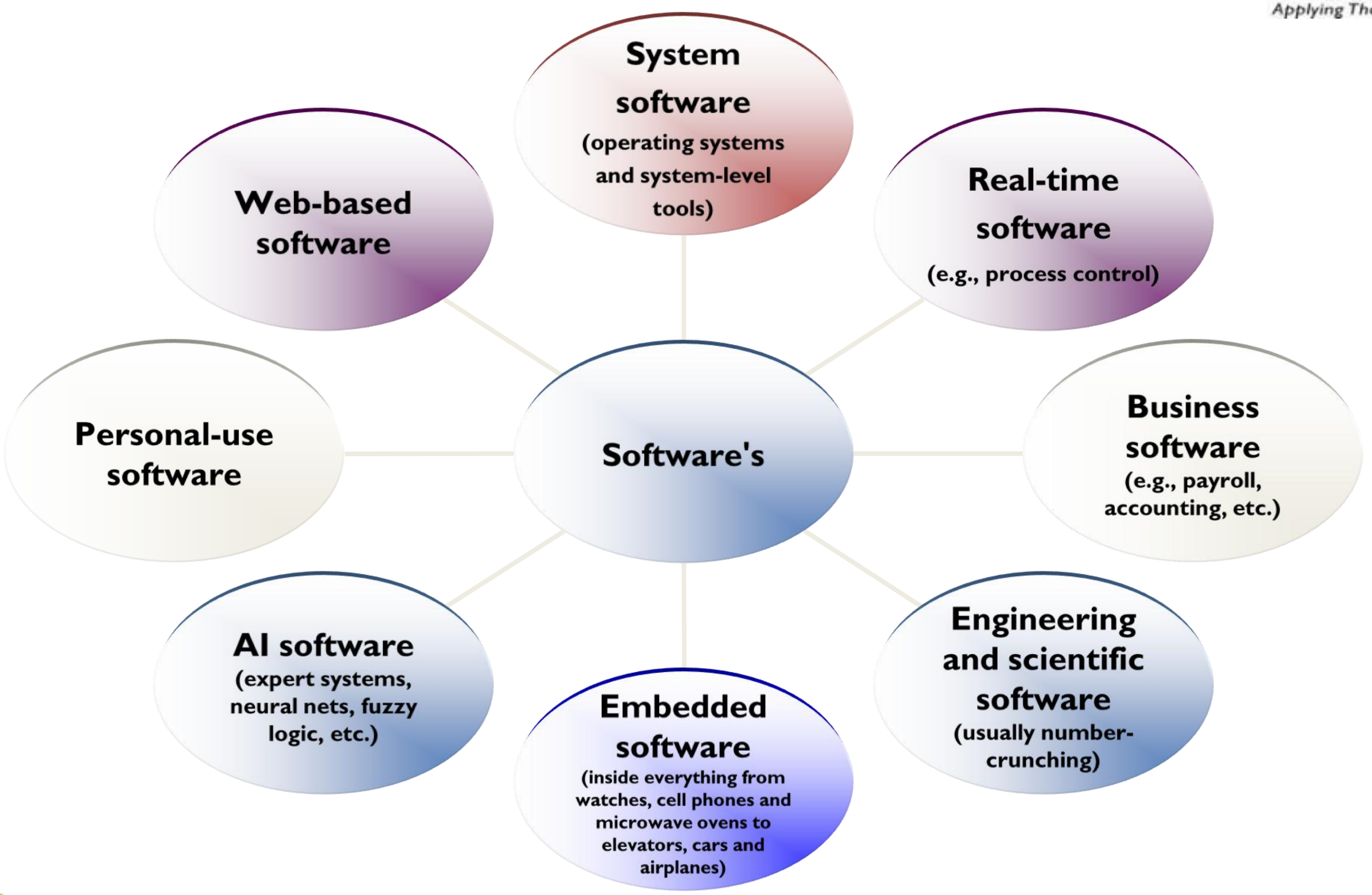
The application of systematic, disciplined and quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

– IEEE

Software Engineering Today

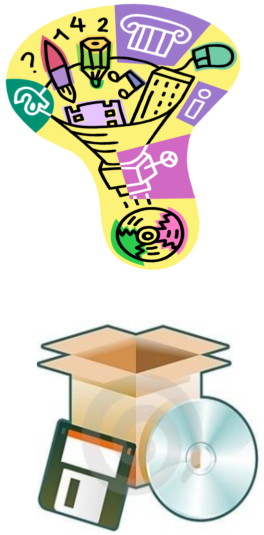


Types of Software


Software's
System software (operating systems and system-level tools)
Real-time software (e.g., process control)
Business software (e.g., payroll, accounting, etc.)
Engineering and scientific software (usually number-crunching)
Embedded software (inside everything from watches, cell phones and microwave ovens to elevators, cars and airplanes)
AI software (expert systems, neural nets, fuzzy logic, etc.)
Personal-use software
Web-based software

Why Software Engineering?

- . ...didn't meet users' requirements
- ...took more time than expected to write
- . ...cost more money than expected
- . ...was unmanageable and difficult to maintain
- . ...just plain didn't work very well



+

Software Engineering Body of Knowledge

Computing Fundamentals	Software Product Engineering	Software Management	Software Domains
Programming Languages	Requirements Engineering	Project Process Management	Artificial Intelligence
Computer Architecture	Software Design	Risk Management	Database Systems
Algorithms and Data Structures	Software Coding	Quality Management	Human-Computer Interaction
Mathematical Foundations	Software Testing	Configuration Management	Numerical & Symbolic Comp.
Operating Systems	Software Ops& Maint	Dev. Process Management	Computer Simulation
		Acquisition Management	Real-Time Systems

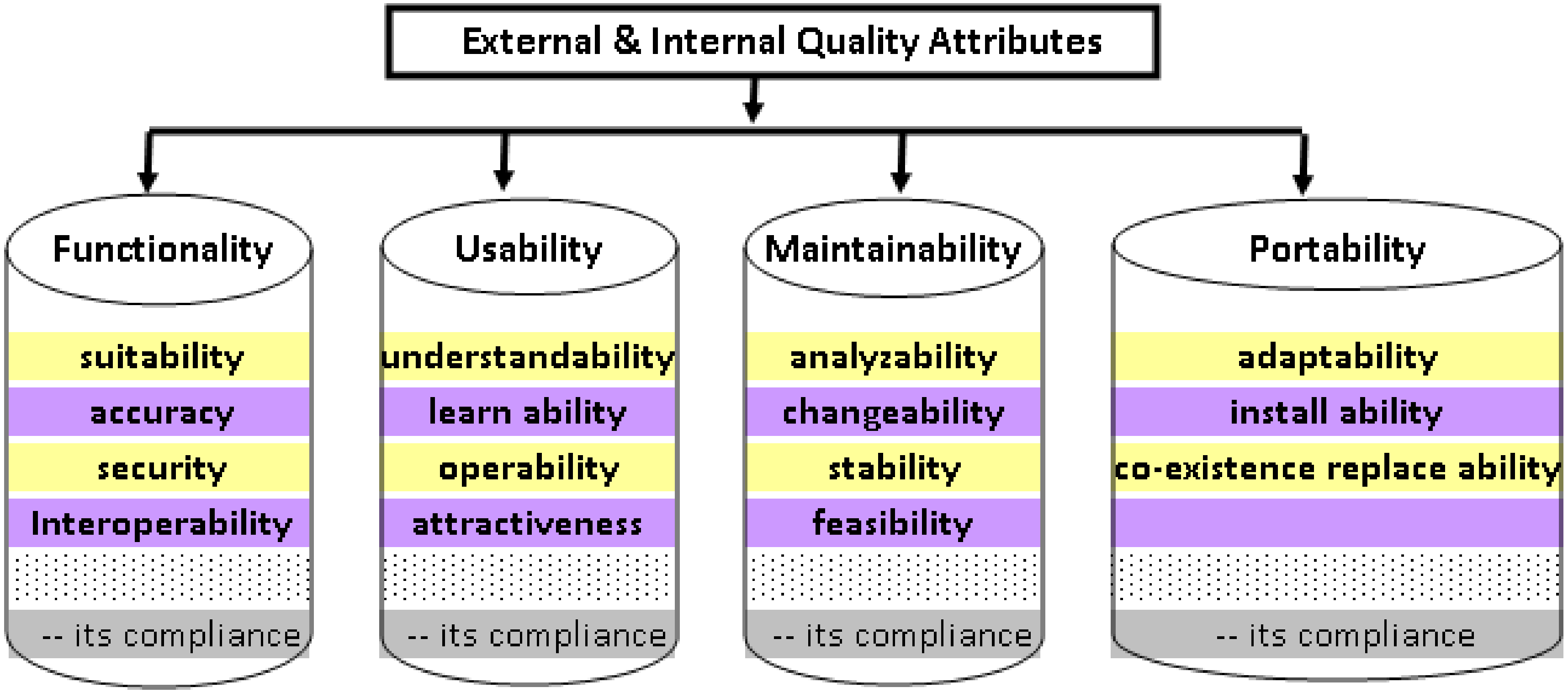
What is good software?

- User view where the quality is fitness for purpose

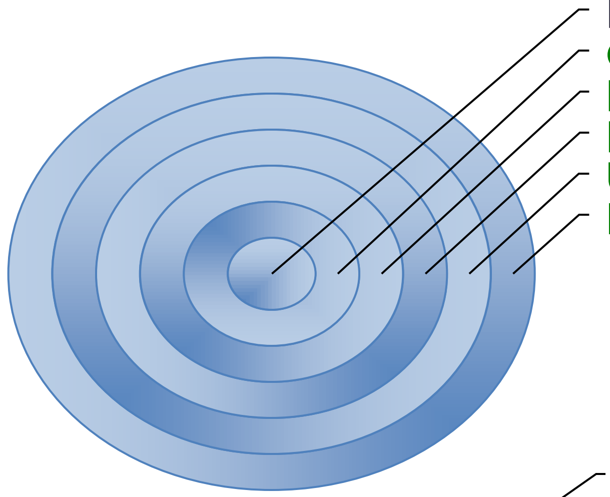
1. The Quality of the Product
2. The Quality of the Process
3. The Quality of the Service
4. The Quality in the Context of Business Environment

Software Quality

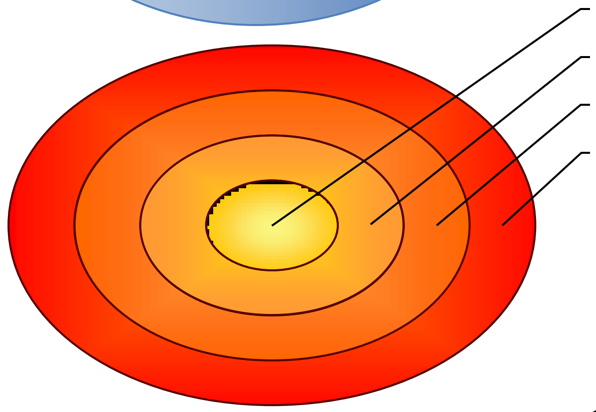
Taxonomy of quality attributes (ISO 9126)



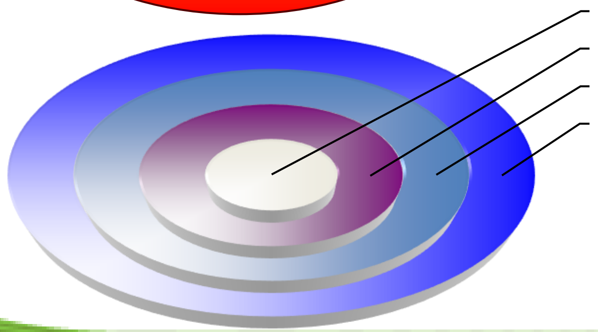
The importance of quality criteria



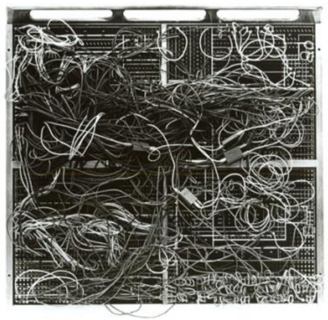
Product operation
Correctness: does it do what I want?
Reliability: does it do it accurately all of the time?
Efficiency: will it run on my hardware as well as it can?
Usability: can I use it?
Integrity: is it secure?



Product transition
Interoperability: will I be able to interface it with another system?
Reusability: will I be able to reuse some of the software?
Portability: will I be able to use it on another machine?



Product revision
Flexibility: can I change it?
Maintainability: can I fix it?
Testability: can I test it?



Software product characteristics

- Successful software...
- ...can be "easily" maintained
- ...provides the required functionality
- ...is usable by real (i.e. naive) users
- ...is predictable, reliable and dependable
- ...functions efficiently
- ...has a "life-time" (measured in years)
- ...provides an appropriate user interface¤
- ...is accompanied by complete documentation
- ...may have different configurations
-

"consumer desires"

- Solves the problem
- Reliable and Available
- Powerful
- Fast and Flexible
- Low cost to buy
- Ability to learn and use

"producer desires"

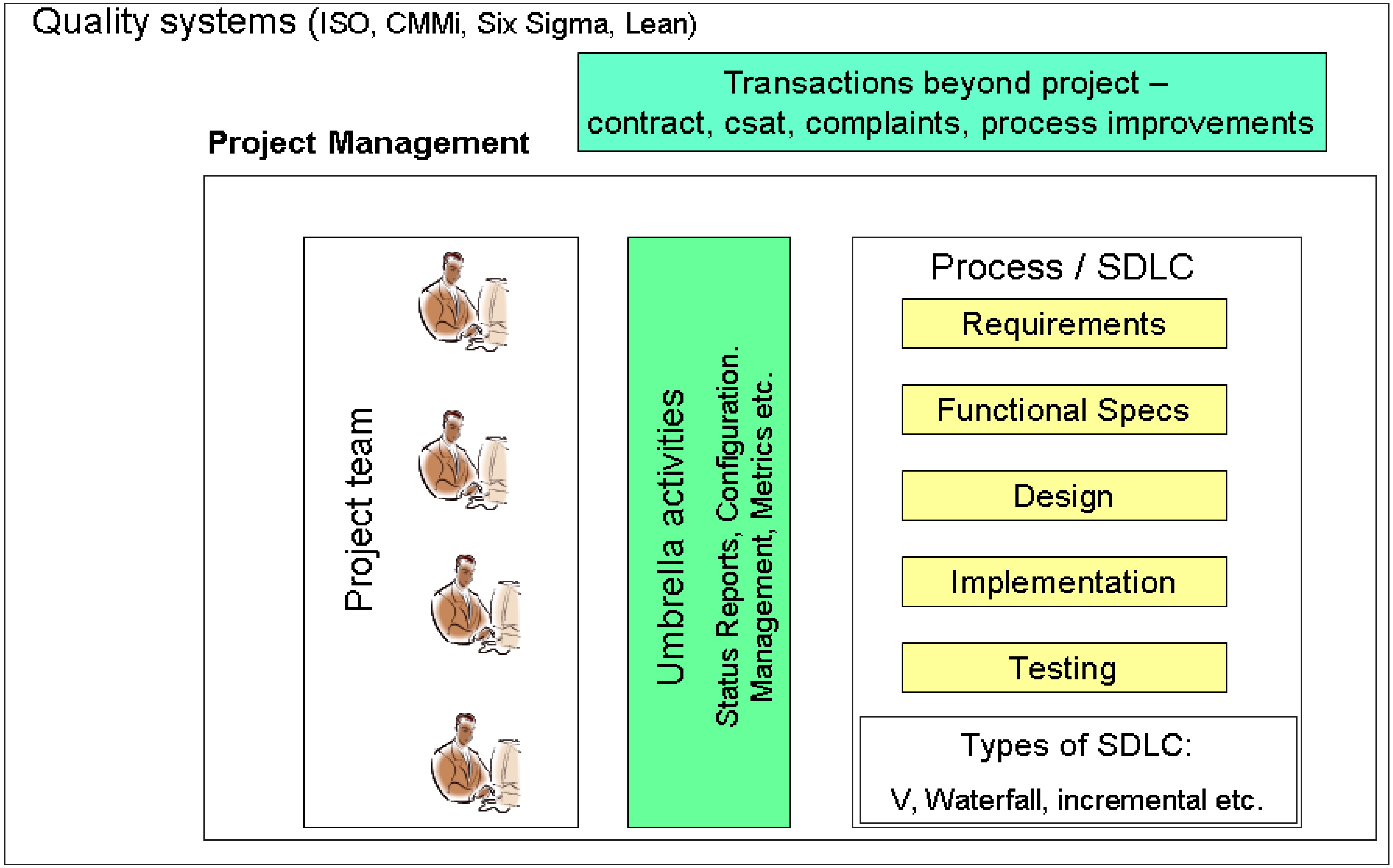
- Low cost to produce
- Well-defined behavior
- Easy to "sell"
- Ability to maintain
- Reliable and Flexible
- Easy to use
- Time to produce

The development of software engineering

Research limelight:

- Specification
- Methodical program development
- Structuring of software systems
- Reusability of software components
- Project organization
- Requirements for software development tools
- Automatic generation of software
- Quality assurance
- Documentation

Software engineering – Bird's eye view


Quality systems (ISO, CMMi, Six Sigma, Lean)
Transactions beyond project – contract, csat, complaints, process improvements
Project Management
Project team
Umbrella activities
Status Reports, Configuration Management, Metrics etc.
Process / SDLC
Requirements
Functional Specs
Design
Implementation
Testing
Types of SDLC:
V, Waterfall, incremental etc.

Summary

In this module, we discussed:

- Software Engineering
- Characteristics of "good software"
- Systems approach to build a software
- Software quality
- Quality attributes

Software Development Models

Module 2

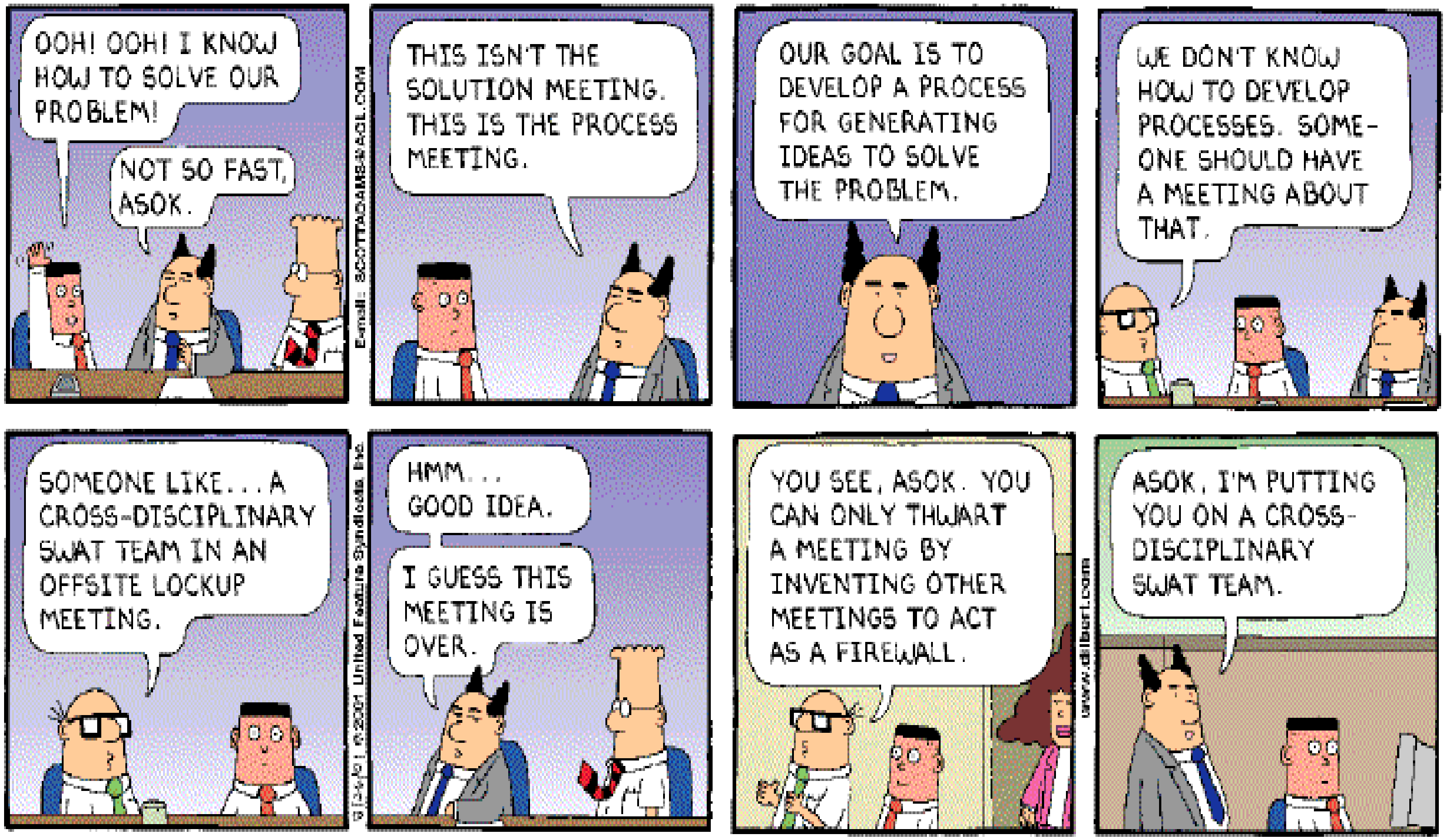
Objectives

At the end of this module, you will be able to:

- Define what is meant by the term “process” and how it applies to software development
- Describe the activities, resources and products involved in software development process
- Describe Waterfall, V, Spiral and Incremental models of software development process and understand their drawbacks in terms of applicability
- Describe Prototype-oriented & Object-oriented models of software development process and understand their drawbacks in terms of applicability
- Recognize IT Project life cycles with roles and responsibility of stake holders in the process
- Describe characteristics of several different tools and techniques for process modeling

Duration: 4 hrs

What is a “Process”?


OOH! OOH! I KNOW HOW TO SOLVE OUR PROBLEM!
NOT SO FAST, ASOK.
THIS ISN'T THE SOLUTION MEETING. THIS IS THE PROCESS MEETING.
OUR GOAL IS TO DEVELOP A PROCESS FOR GENERATING IDEAS TO SOLVE THE PROBLEM.
WE DON'T KNOW HOW TO DEVELOP PROCESSES. SOMEONE SHOULD HAVE A MEETING ABOUT THAT.
SOMEONE LIKE... A CROSS-DISCIPLINARY SWAT TEAM IN AN OFFSITE LOCKUP MEETING.
HMM... GOOD IDEA.
I GUESS THIS MEETING IS OVER.
YOU SEE, ASOK. YOU CAN ONLY THWART A MEETING BY INVENTING OTHER MEETINGS TO ACT AS A FIREWALL.
ASOK, I'M PUTTING YOU ON A CROSS-DISCIPLINARY SWAT TEAM.
E-mail: SCOTTADAMS@AOL.COM
© 2001 United Feature Syndicate, Inc.
www.dilbert.com

What is a "Process" (Contd.).

- Process is a series of steps involving activities constraints and resources that produce an intended output of some kind. - in building a software product; process is refer to a life cycle

- The process prescribes all of the major process activities.
  - Each process activity has entry and exit criteria, i.e. when the activity begins and ends
  - Activities are organized in sequence, hence an indication of when one activity is performed relative to other activities

- Constraints or controls may apply to an activity, or a resource or a product

- Every process has guiding principles that explain goals of each activity

The classical sequential software life-cycle model

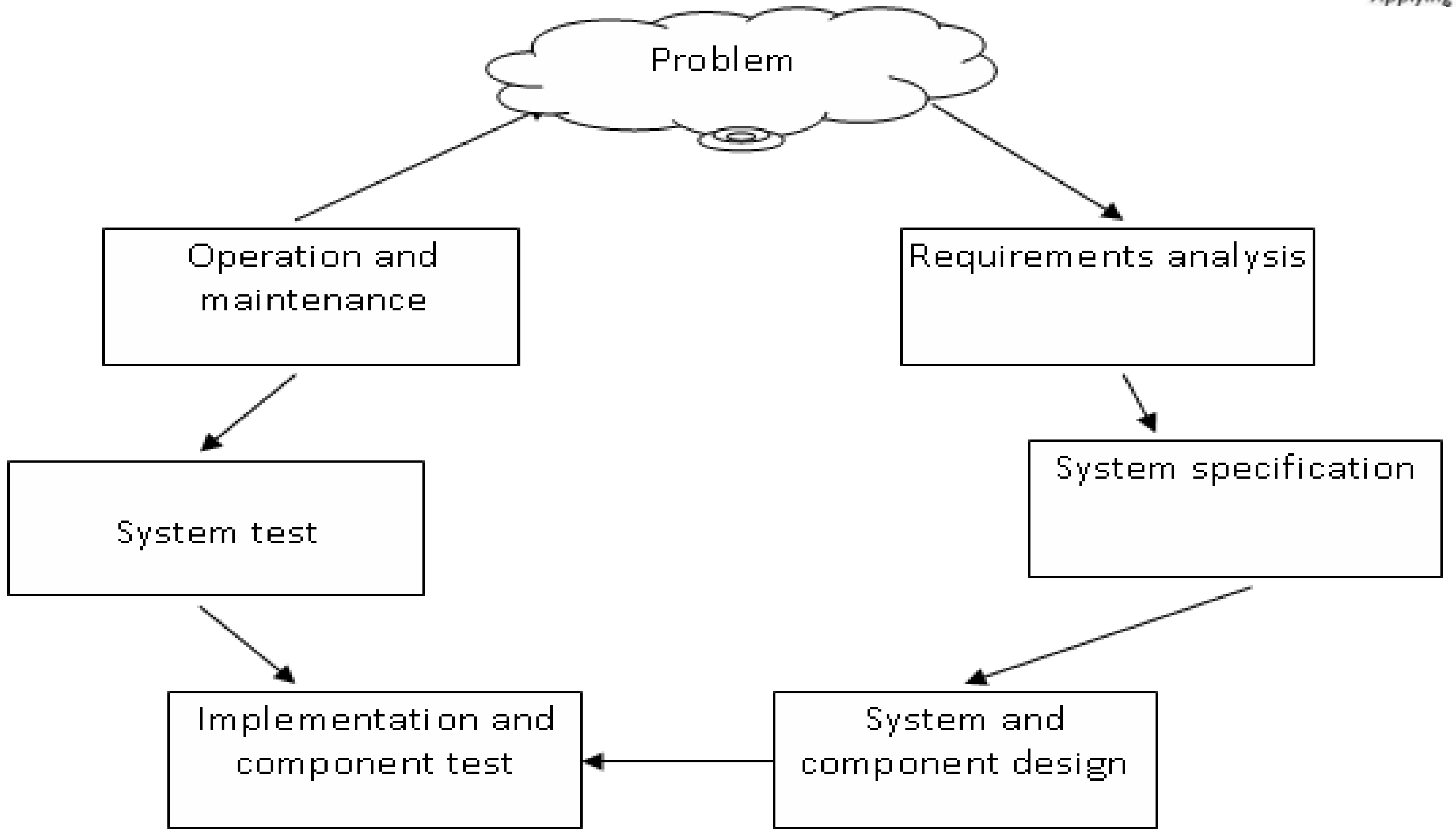
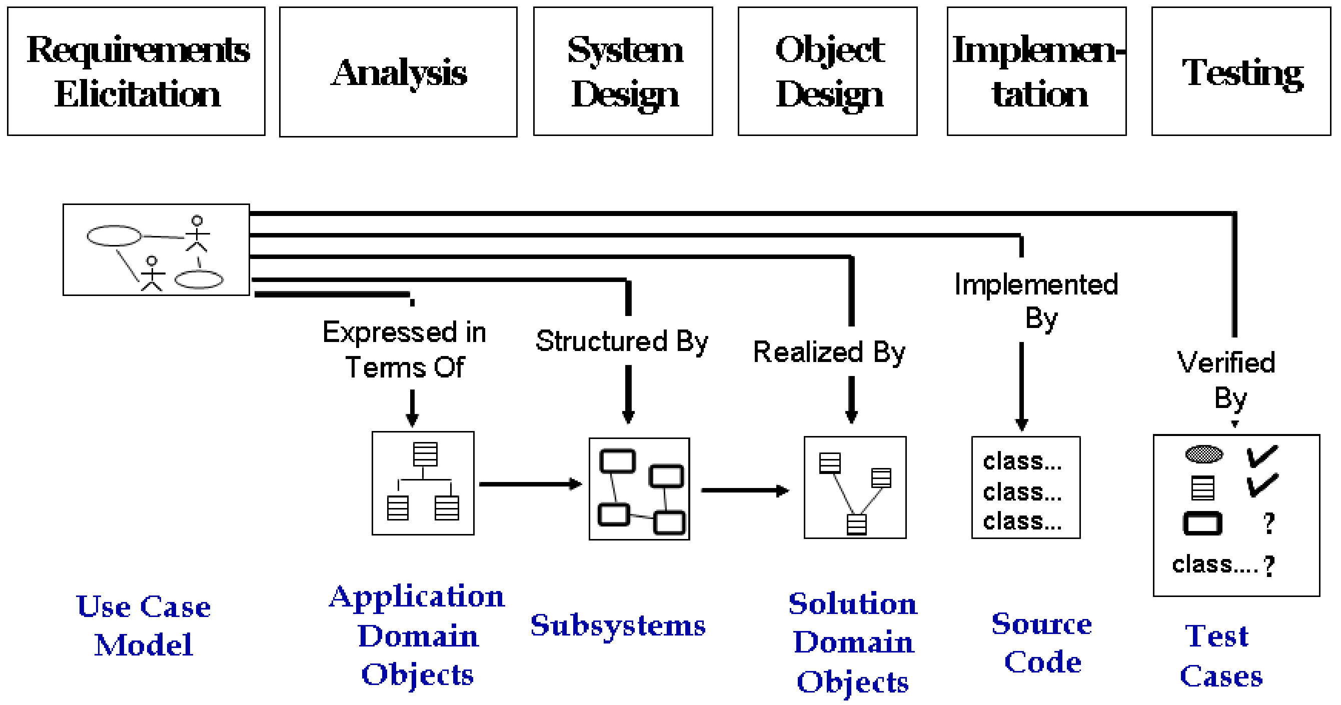
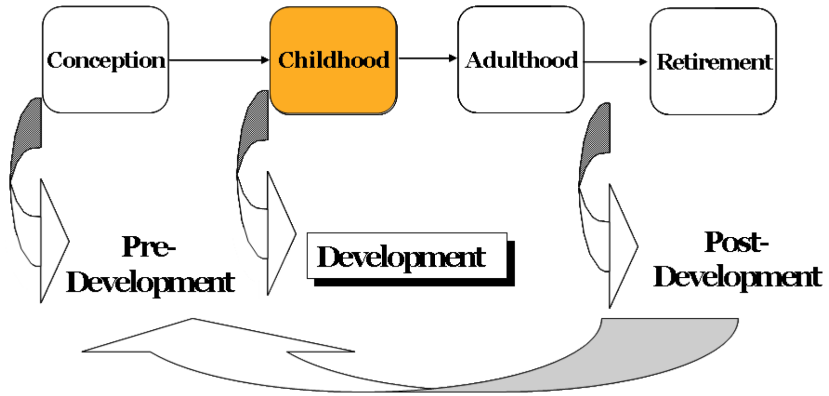

Problem
Requirements analysis
System specification
System and component design
Implementation and component test
System test
Operation and maintenance

Figure: The classical sequential software life-cycle model

Software Lifecycle

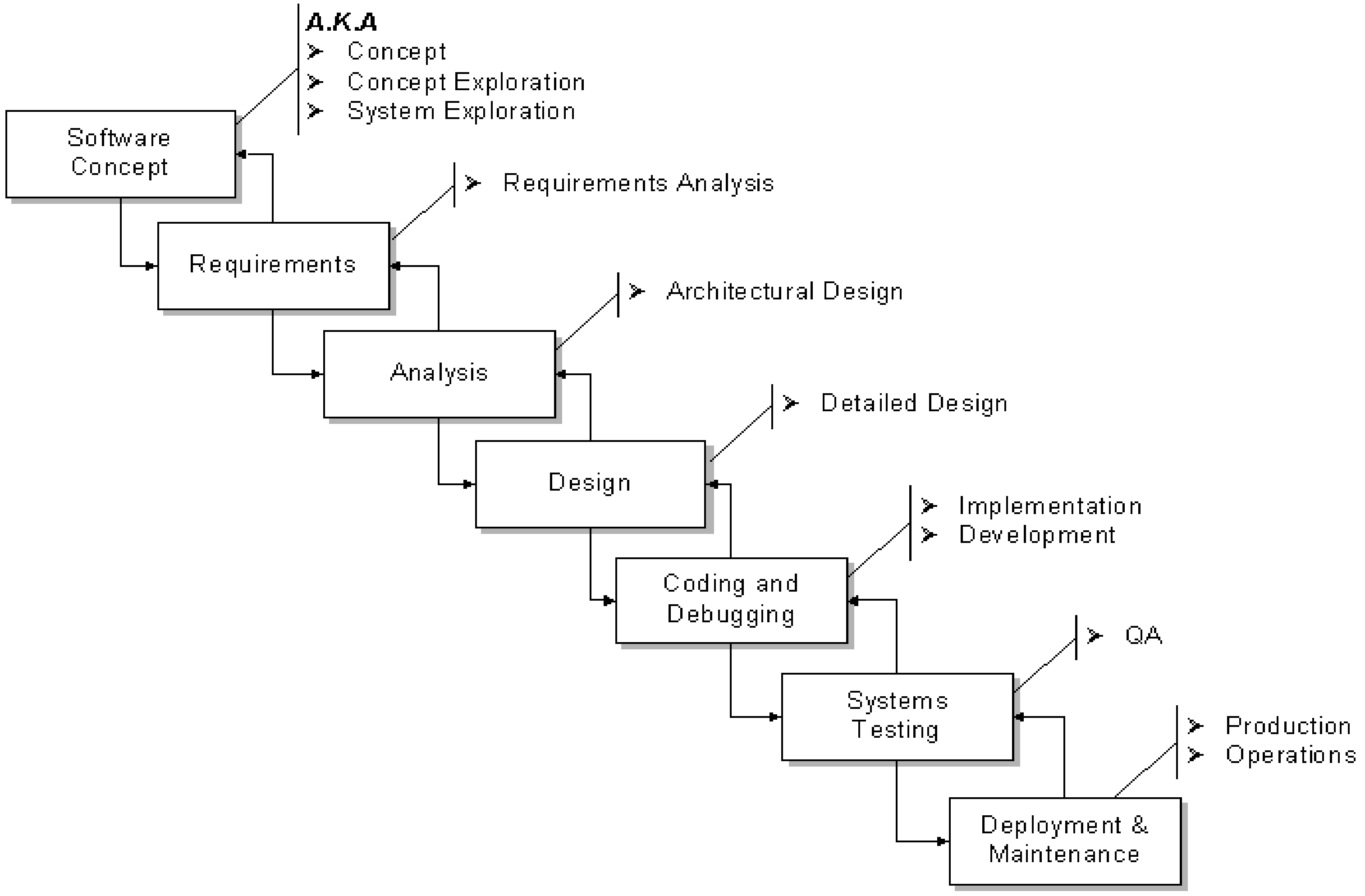




Software Development Lifecycle - Review

Requirements Analysis	**What is the problem?**	**Application Domain**
System Design	**What can be the solution?**	
Detailed Design	**What are the best mechanisms to implement the solution?**	
Program Implementation	**How is the solution formulated?**	**Solution Domain**
Testing	**Has the problem been solved?**	
Delivery	**Can the customer use the solution?**	
Maintenance	**Are enhancements needed?**	

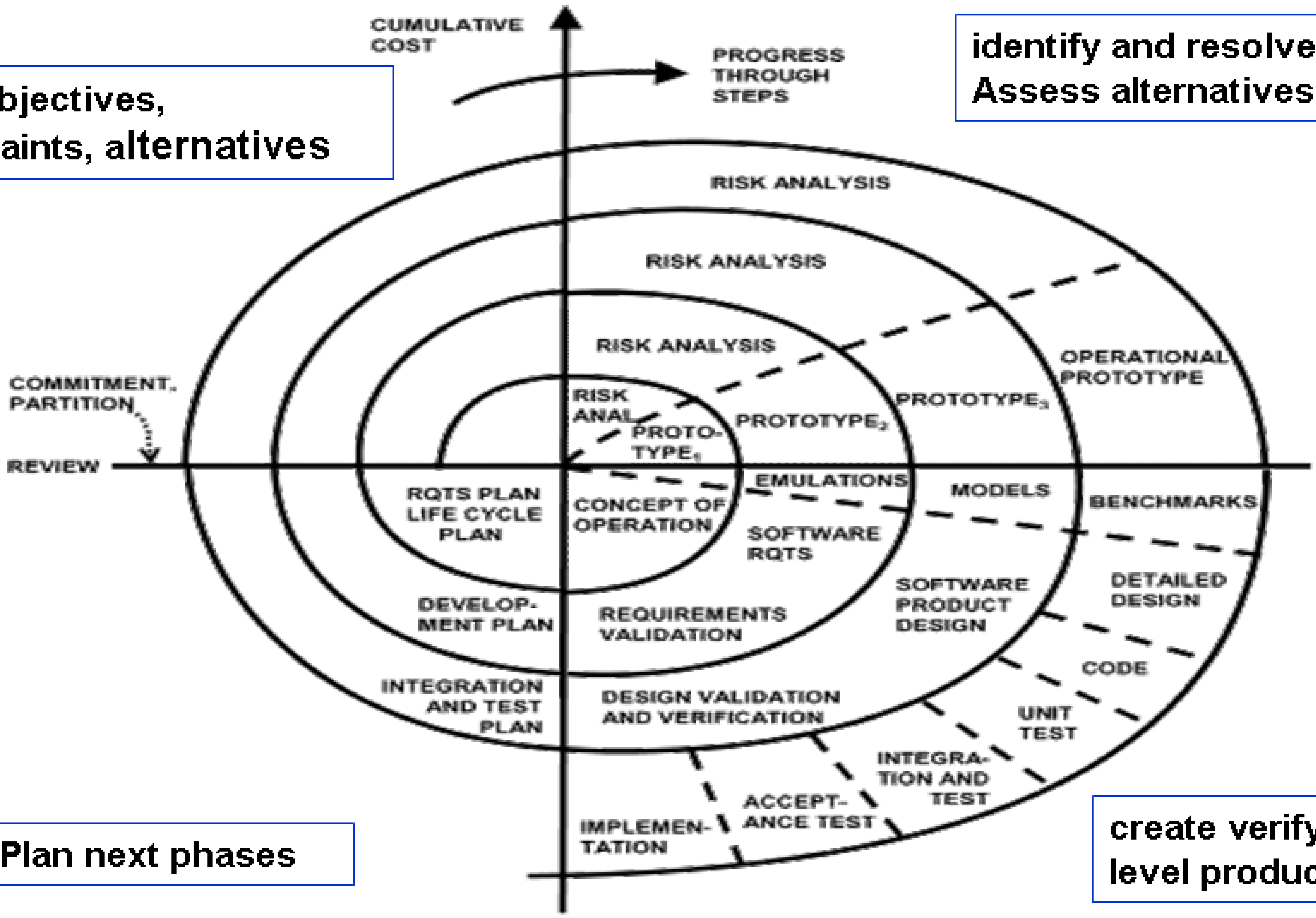
Royce 1970 - The Waterfall model



Boehm 1988 - The Spiral model

Find objectives, constraints, alternatives

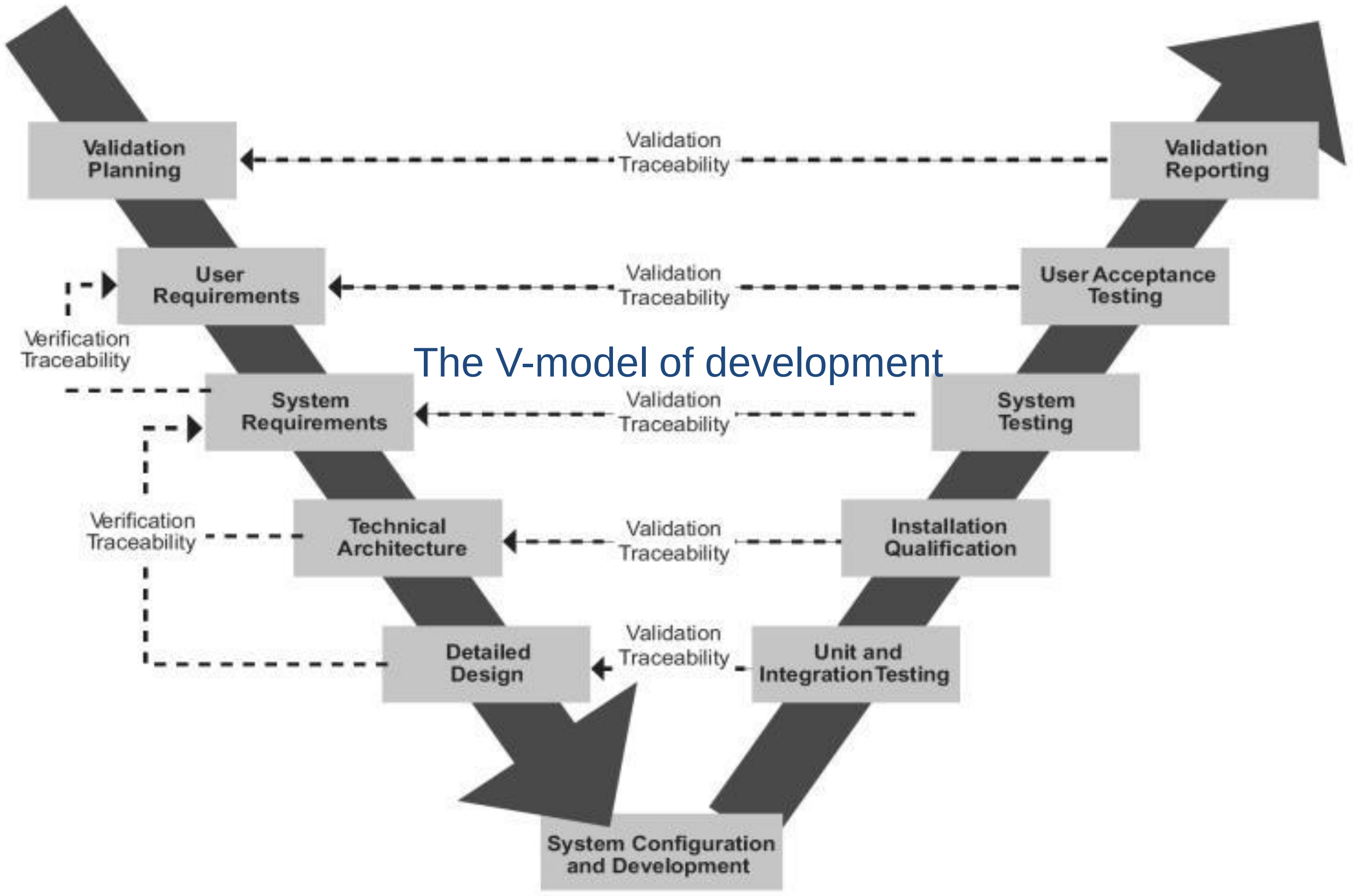
identify and resolve risks; Assess alternatives,



Plan next phases

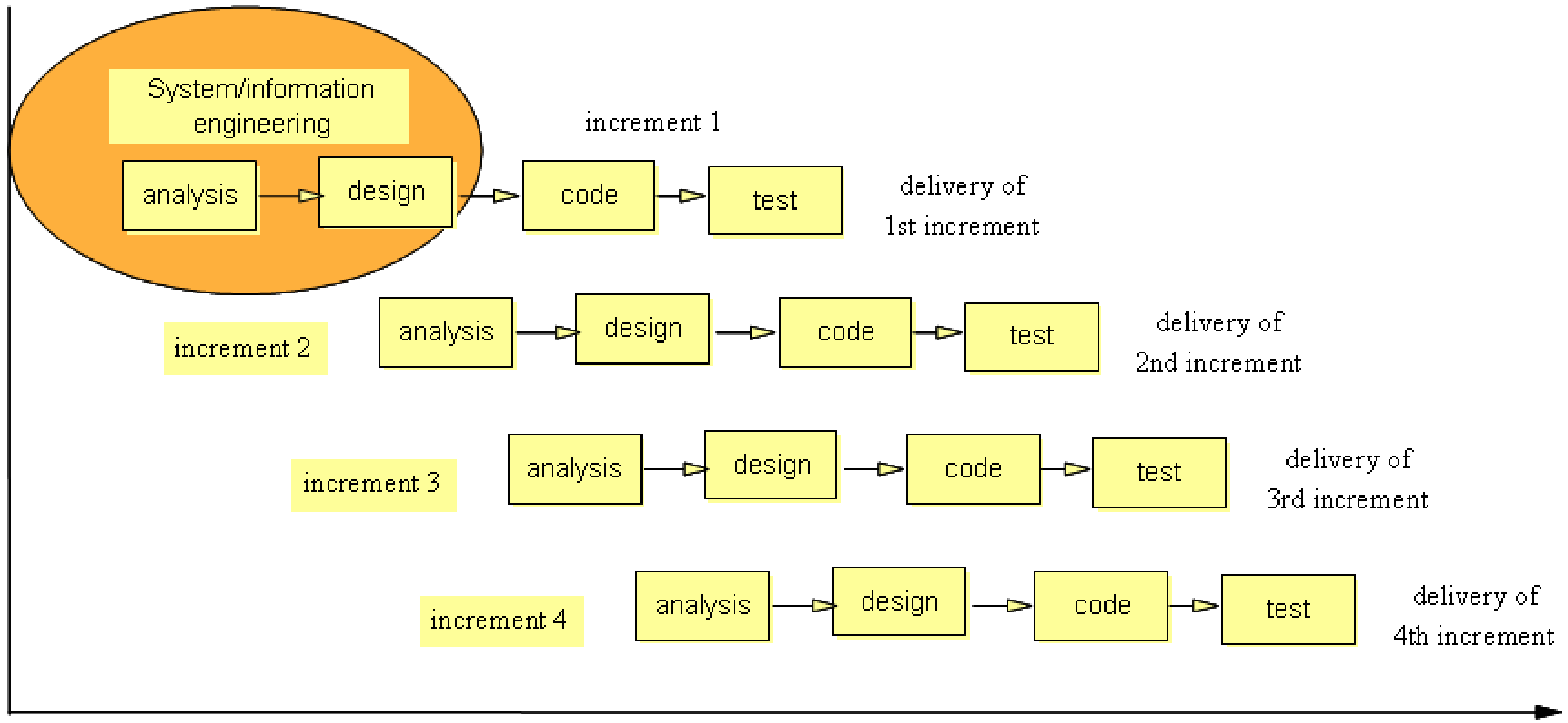
create verify next level product

V Model



The V-model of development

The Incremental Model



The prototyping-oriented model [Pomberger 1991]

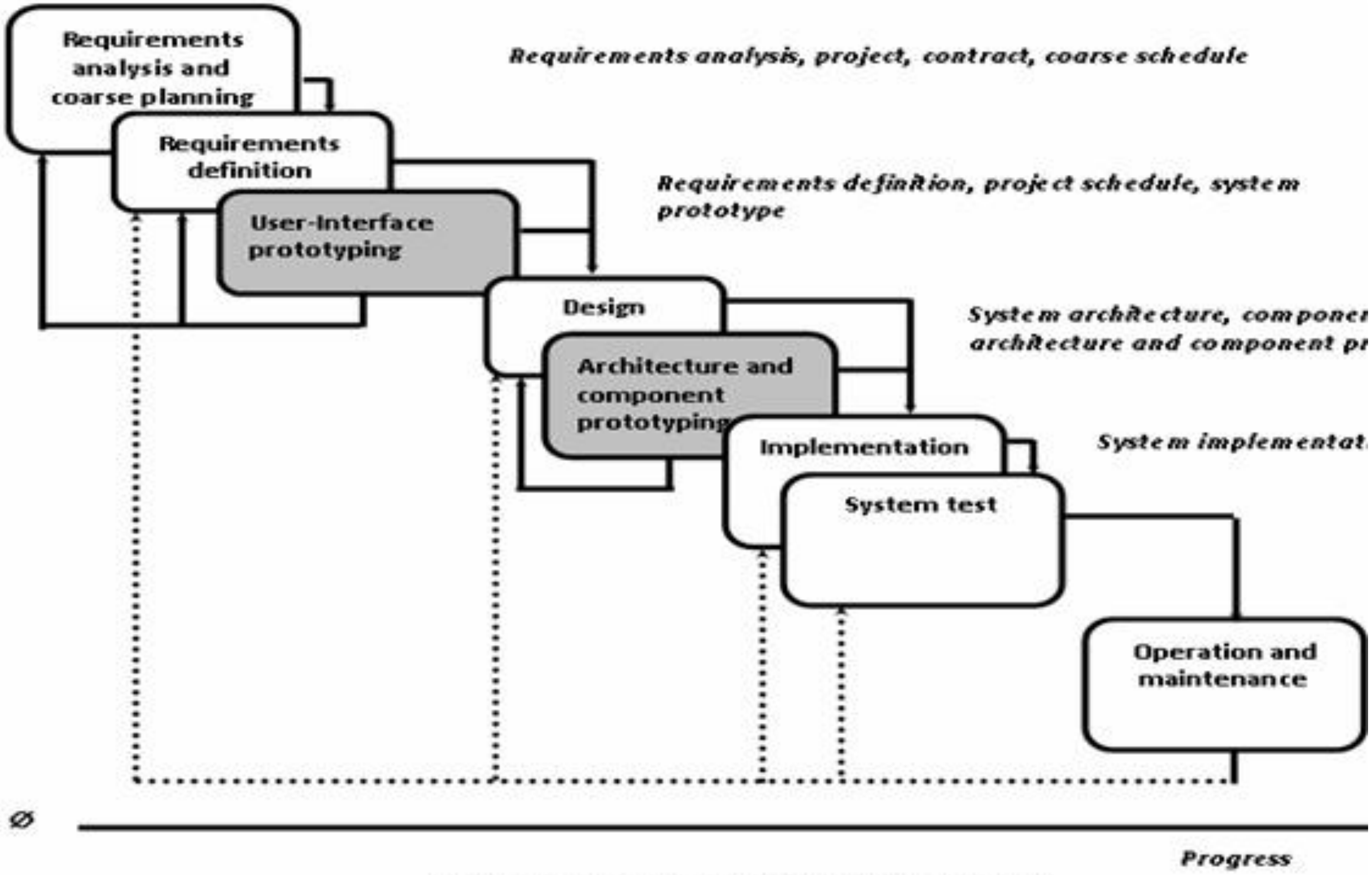
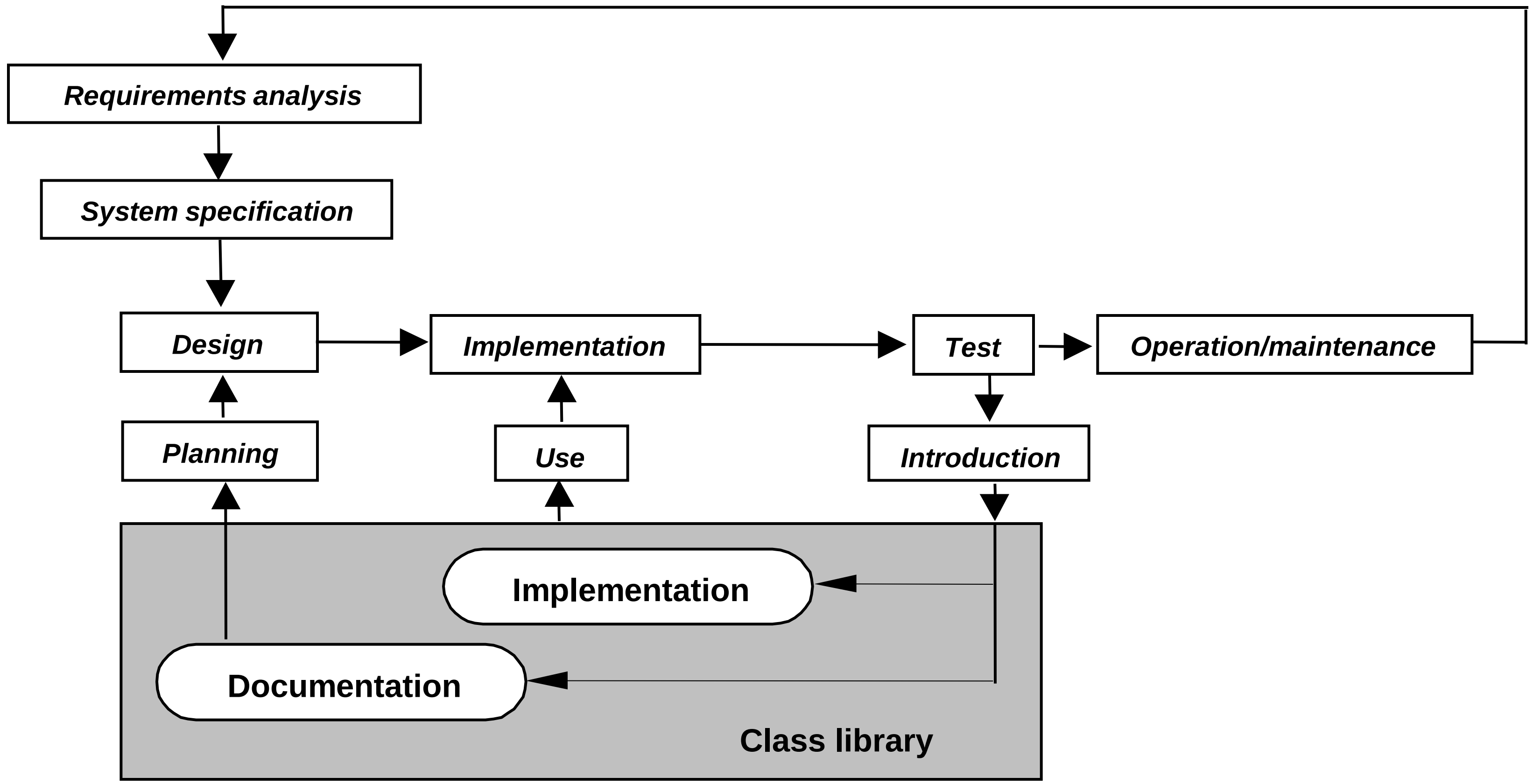
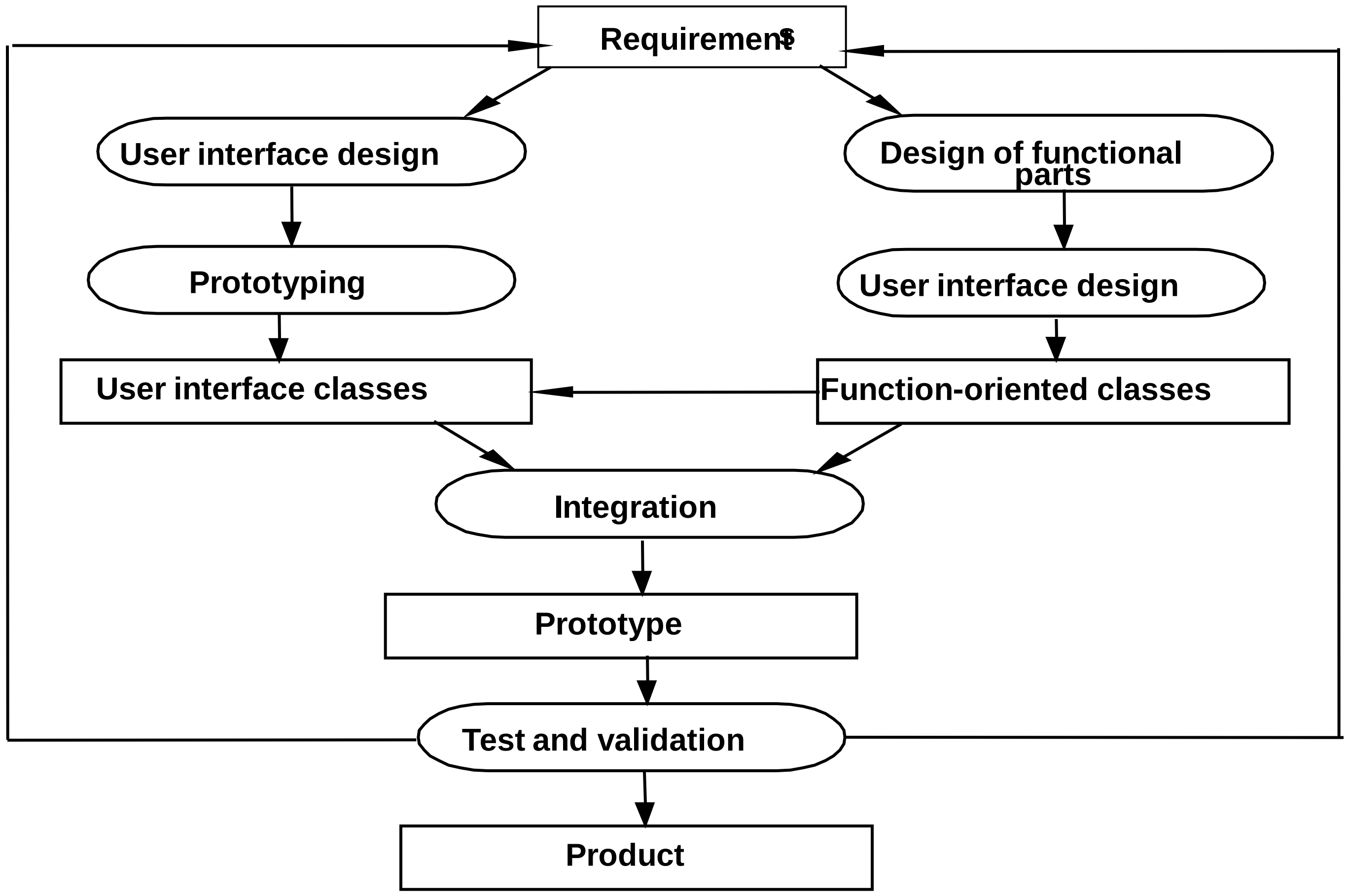

Requirements analysis and coarse planning
Requirements analysis, project, contract, coarse schedule
Requirements definition
Requirements definition, project schedule, system prototype
User-Interface prototyping
Design
System architecture, componen
architecture and component pr
Architecture and component prototyping
Implementation
System implementat
System test
Operation and maintenance
Ø
Progress

Figure: Prototyping-oriented software life cycle

The object-oriented model



The object and prototyping-oriented model

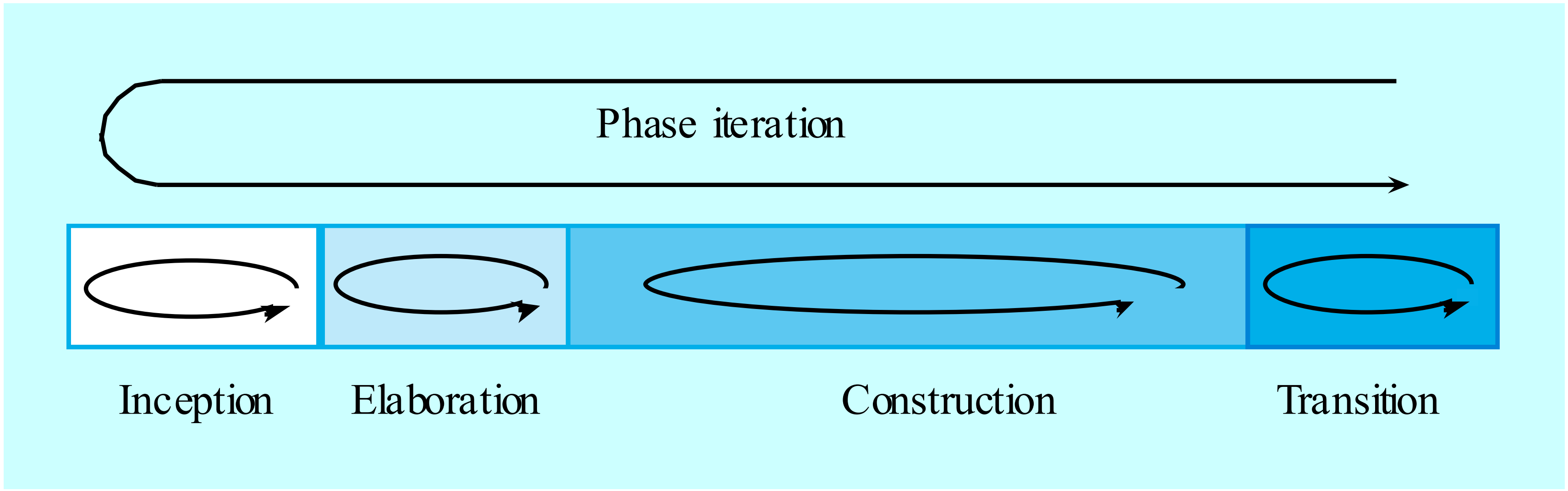


Project life cycles

- IT projects have two concurrent life cycles:
  - Project life cycle (PLC) encompasses all activities of project, including the System Development Life Cycle (SDLC)
  - PLC is directed toward achieving project requirements
  - SDLC provides a direction to achieve product requirements
- Both life cycle models are needed to manage an IT project
  - PLC alone will not adequately address system development concerns
  - SDLC alone will not adequately address business and product integration concerns

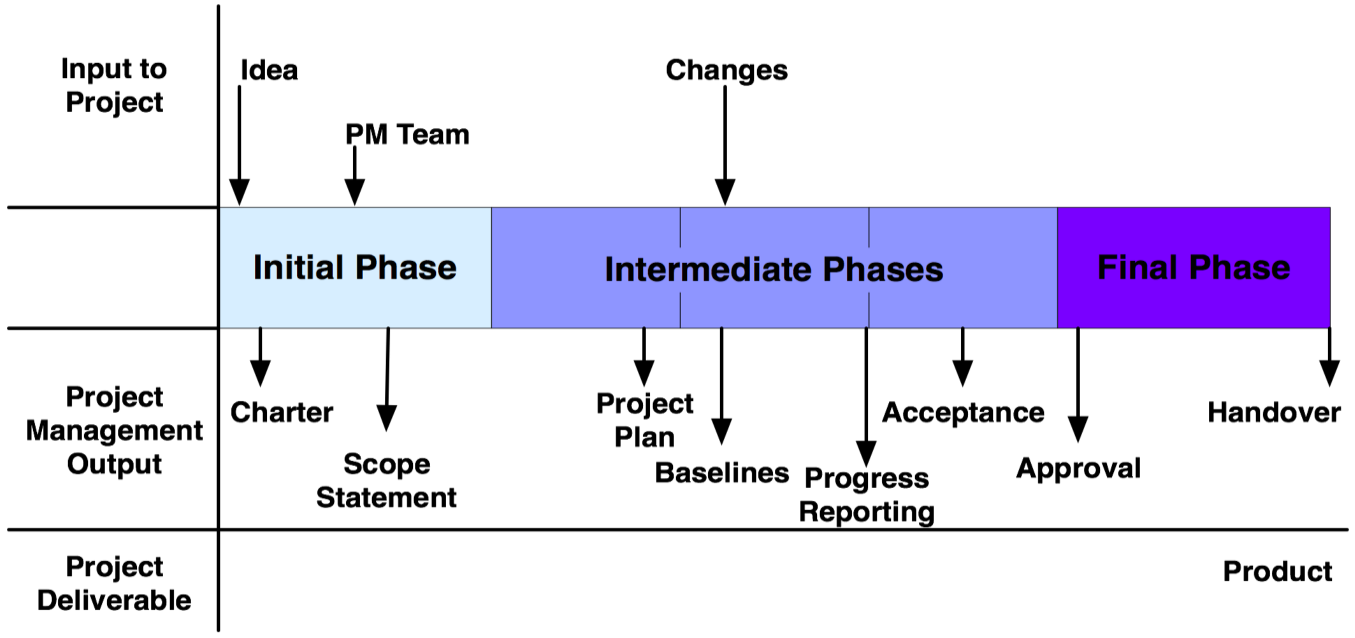
The Rational Unified Process

- A modern process model derived from the work on the UML and associated process
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time
  - A static perspective that shows process activities
  - A proactive perspective that suggests good practice



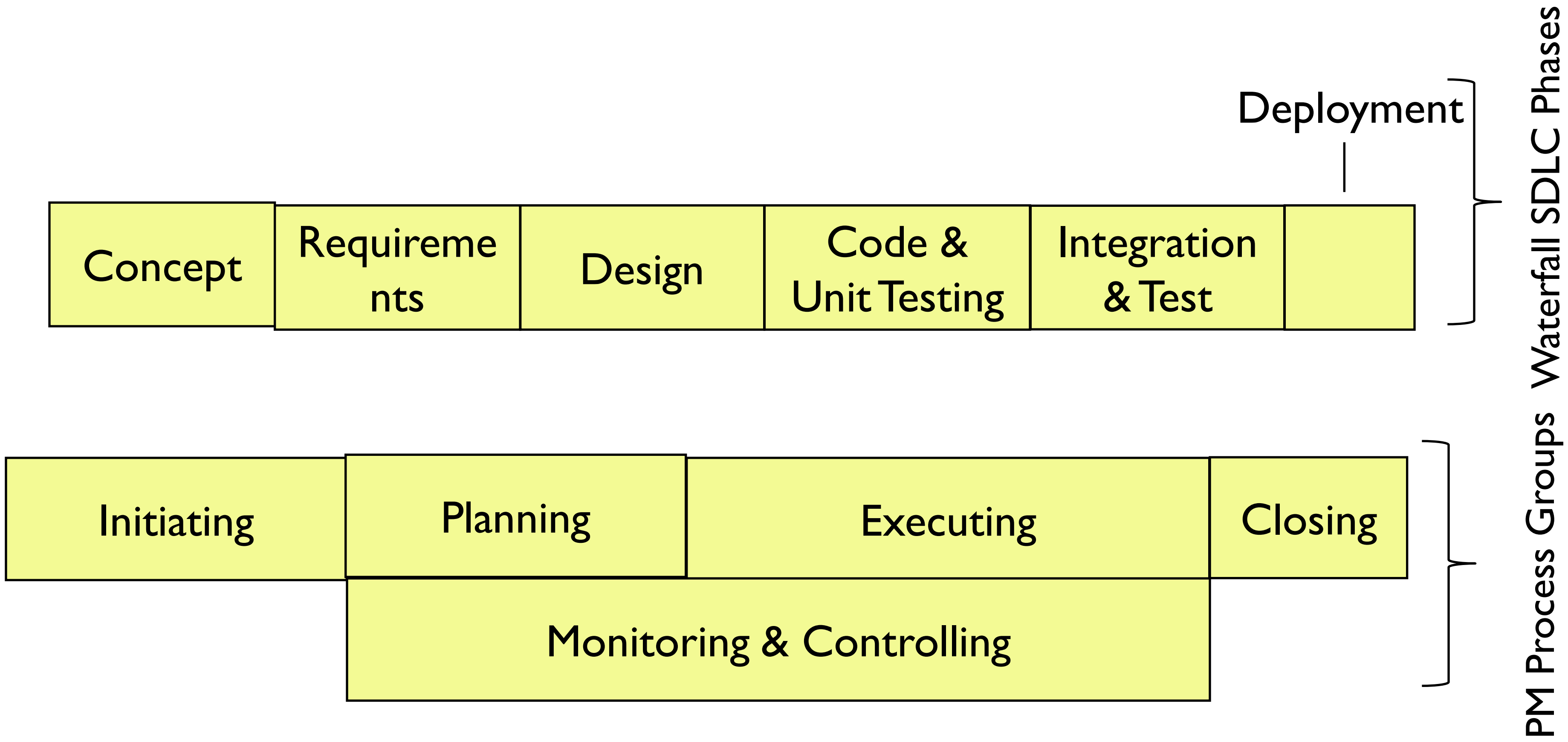
Phases in project life cycle

- The completion and approval of one or more deliverables (measurable, verifiable work product) defines a project phase
- In iterative systems development, new phase can be started without closing the previous phase
- Also a phase can be closed without the initiation of subsequent phase

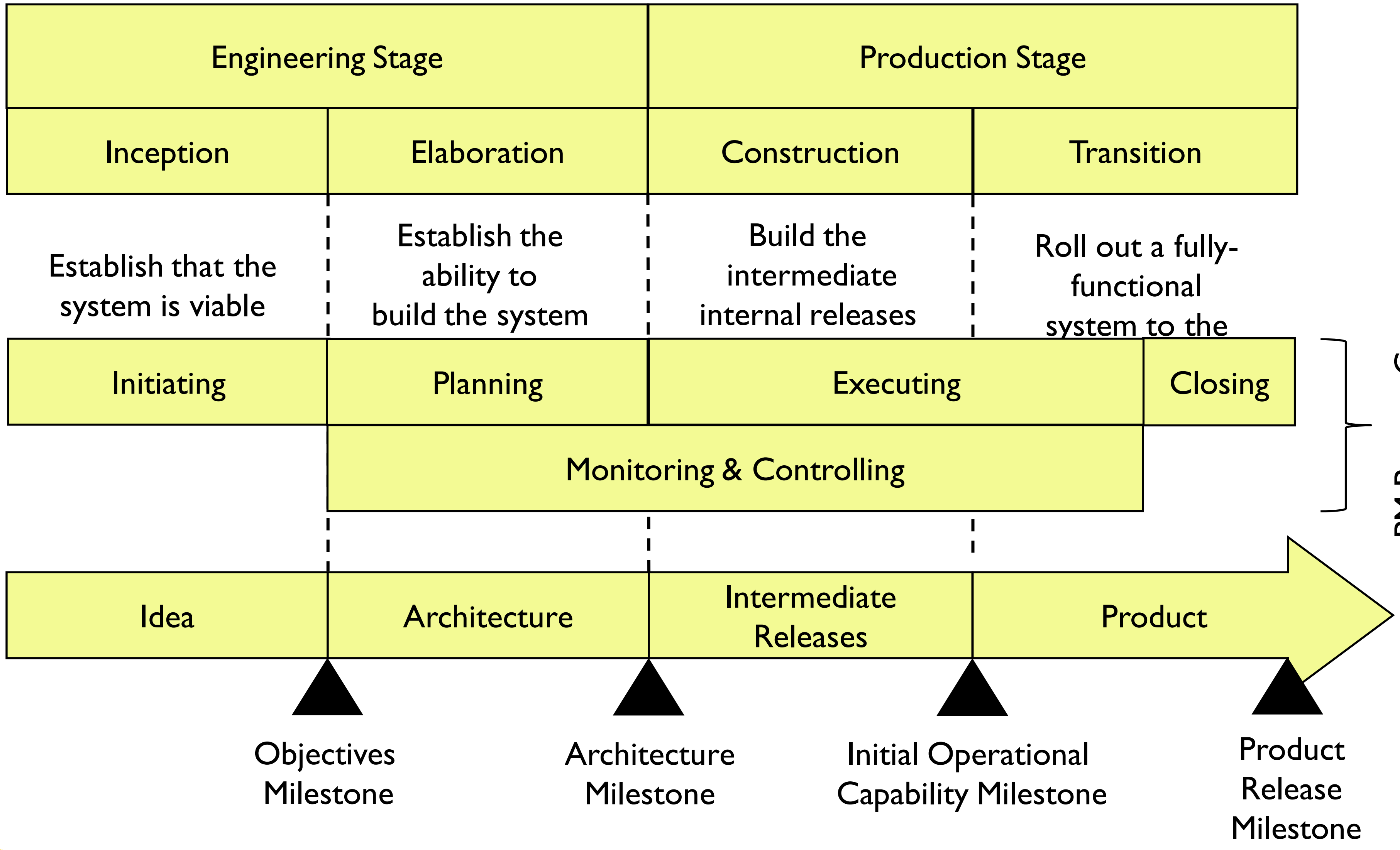


Adapted from PMBOK Guide 3rd Edition Fig. 2-3

Waterfall development model - Project & SDLC integration



Iterative development model - Project & SDLC integration



Summary

In this module, we discussed:

- Process & software development
- Waterfall Model
- Disadvantages of Spiral model
- V Model and Incremental model
- Activities, products and resources involved in software development process
- Various models of the software development process
- Characteristics of several different tools and techniques for process modeling

Requirements & Design

Module 3

Objectives

At the end of this module, you will be able to:

- Explain structure and contents of the requirements phase
- Identify fundamental problems while defining requirements
- Find importance and role of design step in the software process
- Explain classification of decomposition methods in design phase.
- Discuss various design approaches for consideration

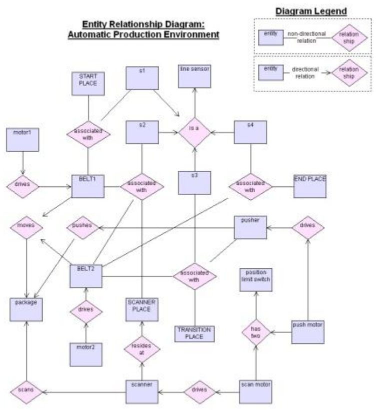
Duration: 2 hrs

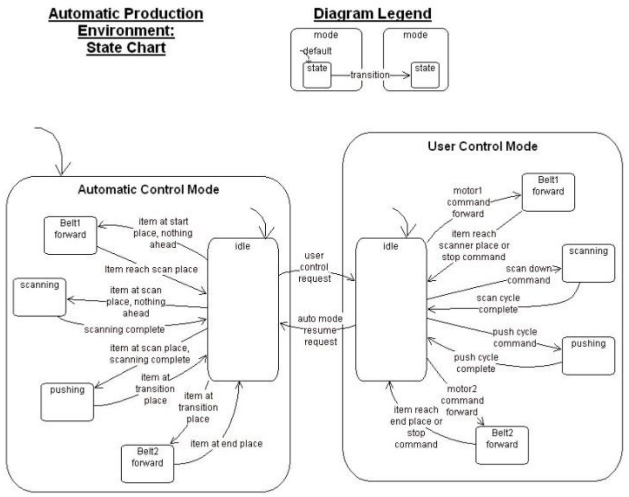
Structure and contents of the requirements definition

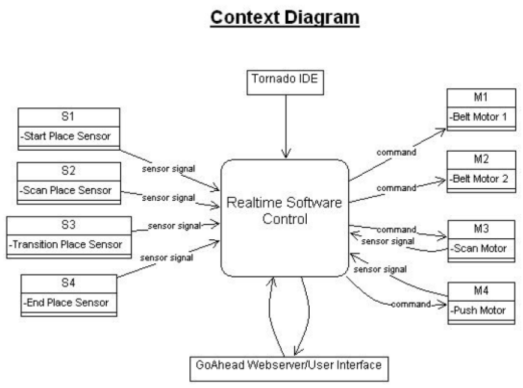
- Initial situation and goals
- System application and system environment
- User interfaces
- Functional requirements
- Nonfunctional requirements
- Exception handling
- Documentation requirements
- Acceptance criteria
- Glossary and index

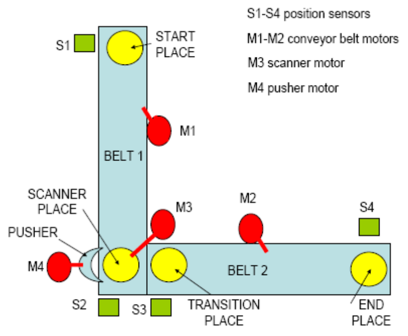
Requirements Phase – An Analysis

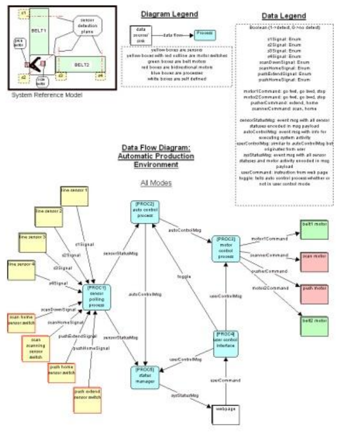
- Data flow diagrams
- Sequence diagrams
- Process specifications
- Entity-relationship diagram
- State charts
- Scenarios
- Use Cases

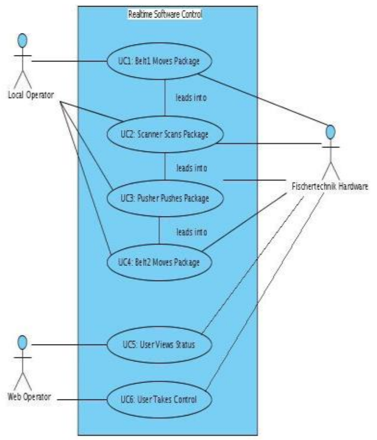










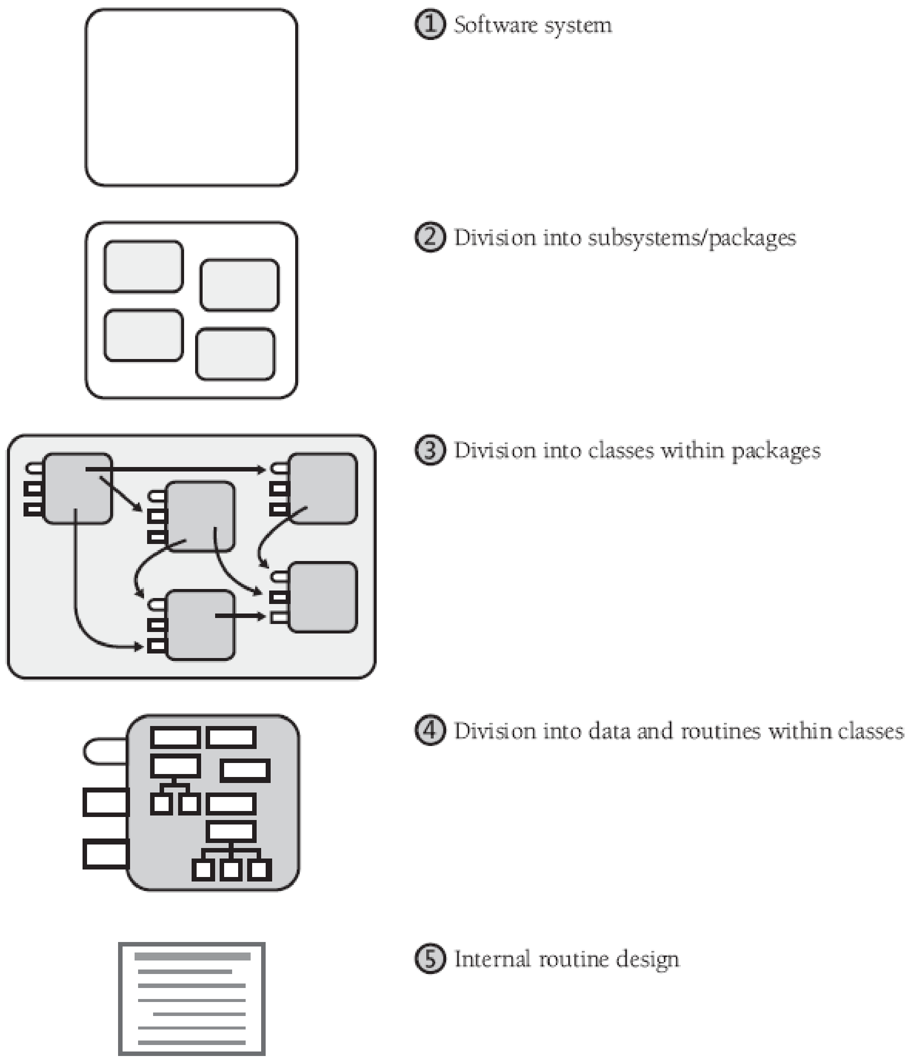


Fundamental problems in defining requirements

- The goal/means conflict
- The purpose and description of functional requirements
- User interfaces designing
- Quality criteria for requirements definition
  - It must be correct and complete
  - It must be consistent and unambiguous
  - It should be minimal
  - It should be readable and comprehensible
  - It must be readily modifiable

Software Design

- Requirements analysis tells us what design specifies how to do it
- What modules will be needed?
- What does each one accomplish?
- What data does each one require?
- How will they interact?



① Software system
② Division into subsystems/packages
③ Division into classes within packages
④ Division into data and routines within classes
⑤ Internal routine design

Software Design process

Two-stage activity:

- Identify the logical design structure, namely the components of a system and their inter-relationships

- Realize this structure in a form which can be executed. This latter stage is sometimes considered as detailed design and sometimes as programming

Design decomposition

Classification of decomposition methods

- Function-oriented decomposition
- Data-oriented decomposition
- Object-oriented decomposition

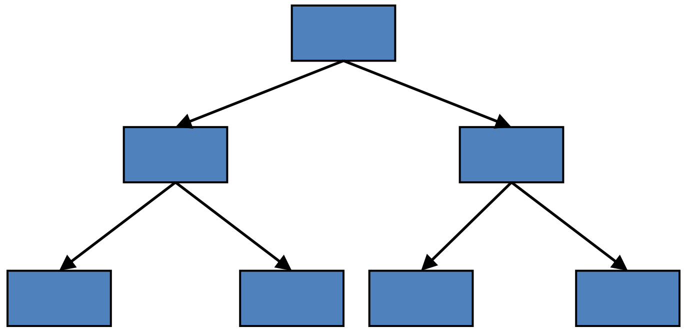
Software Design techniques

- Specification design
  - sits at the technical kernel of the software engineering process
  - is applied regardless of the software process model that is used
  - is the first out of three technical activities such as design, code generation, and testing – usually followed to built and verify the software.

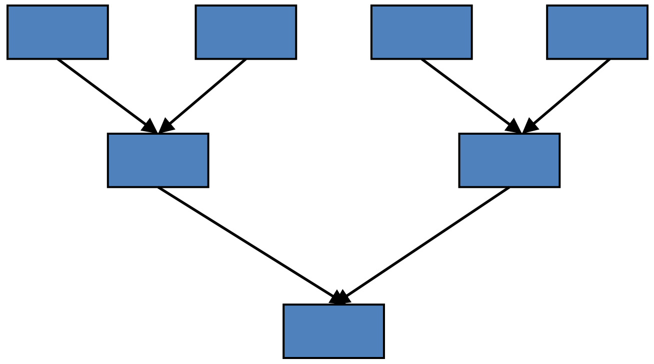
- Principle of abstraction is one of the most important principles for mastering the complexity of software systems.

Design approaches

- **Top-down design**
  - The design activity must begin with the analysis of the requirements definition and implementation details at first, should not be considered



- **Bottom-up design**
  - The fundamental idea: To perceive the hardware and layer above it as an abstract machine



Summary

In this module, we discussed:

- Structure and contents of requirements phase
- Challenges in defining requirements
- Importance and role of design step in the software process
- Classification of decomposition methods considered during design phase
- Various design approaches

Software Implementation

Module 4

Objectives

At the end of this module, you will be able to:

- Identify issues which affect code quality based o implementation
- Find out and discuss the quality concerns with the given code snippets
- Discuss the elements of good programming style with such expressiveness, structure and efficiency
- Discuss the main properties of software such as portability and reuse
- Recognize the importance of coding standards
- Explain the importance and need of refactoring technique during development phase

Duration: 4 hrs

Software Implementation

- Implementation of a software system is equal to the transformation (coding) of the design results into programs that are executable on certain target machine
- A good implementation reflects the design decisions
- A computer has no concept of "well-written" source code
- Programming style guides, which often stress readability and usually language-specific conventions are aimed at reducing the cost of maintenance

Issues that affect code quality

- Readability and Understandability
- Maintainability and Manageability
- Compatibility and Portability
- Performance: resource consumption of memory, CPU
- Robustness

Implementation - Readability

- There are four key points that make a program readable for humans:
  - well-chosen identifier names
  - consistent indentation
  - good use of horizontal and vertical white space
  - good use of comments

How Not To Do It:

```
void calc (double m[],char *g){
double tm=0.0; for (int t=0;t<MAX_TASKS;t++) tm+=m[t];
int i=int(floor(12.0*(tm-MIN)/(MAX-MIN)));
strcpy(g,let[i]);}
```

One possible better way:

```
void calculateGrade (double marks[], char *grade)
{
  double totalMark = 0.0;

  for (int task = 0; task < MAX_TASKS; task++)
    totalMark += marks[task];

  int deltaT = totalMark - minMarks;
  int deltaM =  maxMarks - minMarks;

  int gradeIndex = int(floor(12.0 * deltaT / deltaM));

  strcpy(grade, letterGrades[gradeIndex]);
}
```

About Comments

- Comments can help readability immensely — but poor comments can harm too
- Good comments
  - agree with the code
    (question: if comments and code **don't** agree, which is wrong?)
  - are written in terms of the problem
  - are non-trivial
- For example, not recommended to do this:

  x = 0;   // set x to zero

  Instead, do something like this:

  x = 0;   // reset the counter (use 'x' with appropriate naming)

```
int np;       // Number of pixels counted.
int flag;     // 1 if there is more input, otherwise 0.
int state;    // 0 = closed, 1 = ajar, 2 = open.
double xcm;   // X-coordinate of center of mass.
double ycm;   // Y-coordinate of center of mass.
```

Quality criteria for programming languages

- Modularity
- Documentation value
- Control flow
- Data structures
- Efficiency
- Portability
- Integrity
- Dialog support
- Specialized language elements

Characteristics of programming languages

- Quality of the compiler
- Company policy
- Availability of libraries
- External requirements
- Availability of development tools

Programming style

- Structure
  - Structuring in the large
  - Structuring in the small
- Expressiveness
  - Expressive power
  - The rules for writing comments
- Outward form
- Efficiency

Portability and reuse

- Software portability
- Machine Architecture dependencies
- Operating system dependencies
- Software reuse

Code Standards

- Why?
  - Gives **less** defects.
  - Code to be **read**, not only written. Several people may **work on and understand** the same code.
  - **Easier/cheaper** maintenance.

- General rules:
  - **Simplicity** – Build simple classes and methods.
  - **Clarity** – Explain where, when, why, and how to use each.
  - **Completeness** – Create complete documentation;
  - **Consistency** – Similar entities should look and behave the same; dissimilar entities should look and behave differently.
  - **Robustness** – Provide predictable documented behaviour in response to errors and exceptions. Avoid hiding errors and do not force clients to detect errors.

Comments; Why, when, where, what

- **Why**
  - To be able to find out what a operation does after a half, one or two years. Automatic API documentation

- **When**
  - Document your code before or when you write it; Design before you implement. Put the design in the operation

- **Where**
  - Before the operation, at specific formulas, decision points etc

- **What**
  - Document the algorithm, avoid unnecessary comments. Refer to a specification if existing

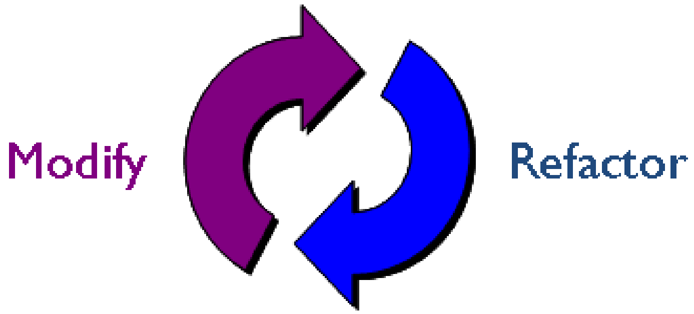
What is Refactoring ?

- **The process of** changing a software system **in such a way that it** does not alter the external behavior **of the code, yet** improves its internal structure.

- "Refactoring - Improving the design of existing code",
  --- Fowler, Refactoring, 1999.

- Refactor is: **to restructure software** by applying a series of refactorings.
  - ...in a series of small, semantics-preserving transformations (i.e. the code keeps working)...
  - ...in order to make the code easier to maintain and modify

- Goals:
  - Better readability, comprehensibility
  - Better design, maintainability and reusability



Why Refactoring?

- Developers are usually concerned with getting the program to work, not about future enhancement
- Increases comprehension of existing code, So that helps during maintenance
- Increased readability leads to the discovery of possible errors

Improves the design of software	the design of a program will decay over time as people change code to realize short-term goal
Makes software easier to understand	by refactoring, software gets cleaned up
Helps you to find bugs	clarifying the structure exposes bugs
Helps you to program faster	as a result from the previous three points
Make code more maintainable	• Easier to read and comprehend and code becomes more reliable with reuse • Variables and functions have names that define behavior

Summary

In this module, we discussed:

- Attributes of programming languages
- Good programming style
  - structure, expressiveness, outward form and efficiency
- Properties of software
  - portability and reuse

Software Testing

Module 5

Objectives

At the end of this module, you will be able to:

- Distinguish between Validation, Verification and Testing
- Recognize categorization of software testing
- Define unit testing, integration testing, System testing
- Describe several different testing strategies and understand their differences
- Classify
  - function testing
  - acceptance testing or installation testing.
  - performance testing

Duration: 3 hrs

Why Software Testing

- Testing is the most authoritative way of verifying software quality
- The system (its business logic, interface and protection mechanisms) is tested with prepared data and scenarios
- Software engineering distinguishes between several types of tests:

Test Type	Description
Conformance or Type	Tests that determine whether a system meets a specified standard.
Functional or System	Tests that determine whether the complete (integrated) system meets its specified requirements.
Unit	Tests that determine whether an individual unit of the source code (e.g. an individual method) works correctly.
Performance	Tests that determine how fast a specific aspect of a system performs under a particular workload.
Load	Tests that determine the usage of the system by simulating multiple users accessing it simultaneously.
Compatibility	Tests that determine whether the application (system) is compatible with the rest of the computer environment (hardware, network, other software, etc).
Portability	Tests that determine portability across system platforms and environments.
Integration	Tests that determine whether individual modules (or a combination of several modules) work correctly. Integration tests are performed after unit tests and before functional tests.

Most Common Software problems

- Incorrect calculation
- Incorrect data edits & ineffective data edits
- Incorrect matching and merging of data
- Data searches that yields incorrect results
- Incorrect processing of data relationship
- Inadequate software performance
- Software usability by end users & Obsolete Software
- Unreliable results or performance
- Inadequate support of business needs
- Incorrect or inadequate interfaces with other systems
- Inadequate performance and security controls

Objectives of testing

- Executing a program with the intent of finding an error
- To check if the system meets the requirements and be executed successfully in the Intended environment
- To check if the system is " Fit for purpose"
- To check if the system does what it is expected to do

- A good test case is one that has a probability of finding an as yet undiscovered error
- A good test is not redundant
- A good test should be "best of breed"
- A good test should neither be too simple nor too complex
- A successful test is one that uncovers a yet undiscovered error

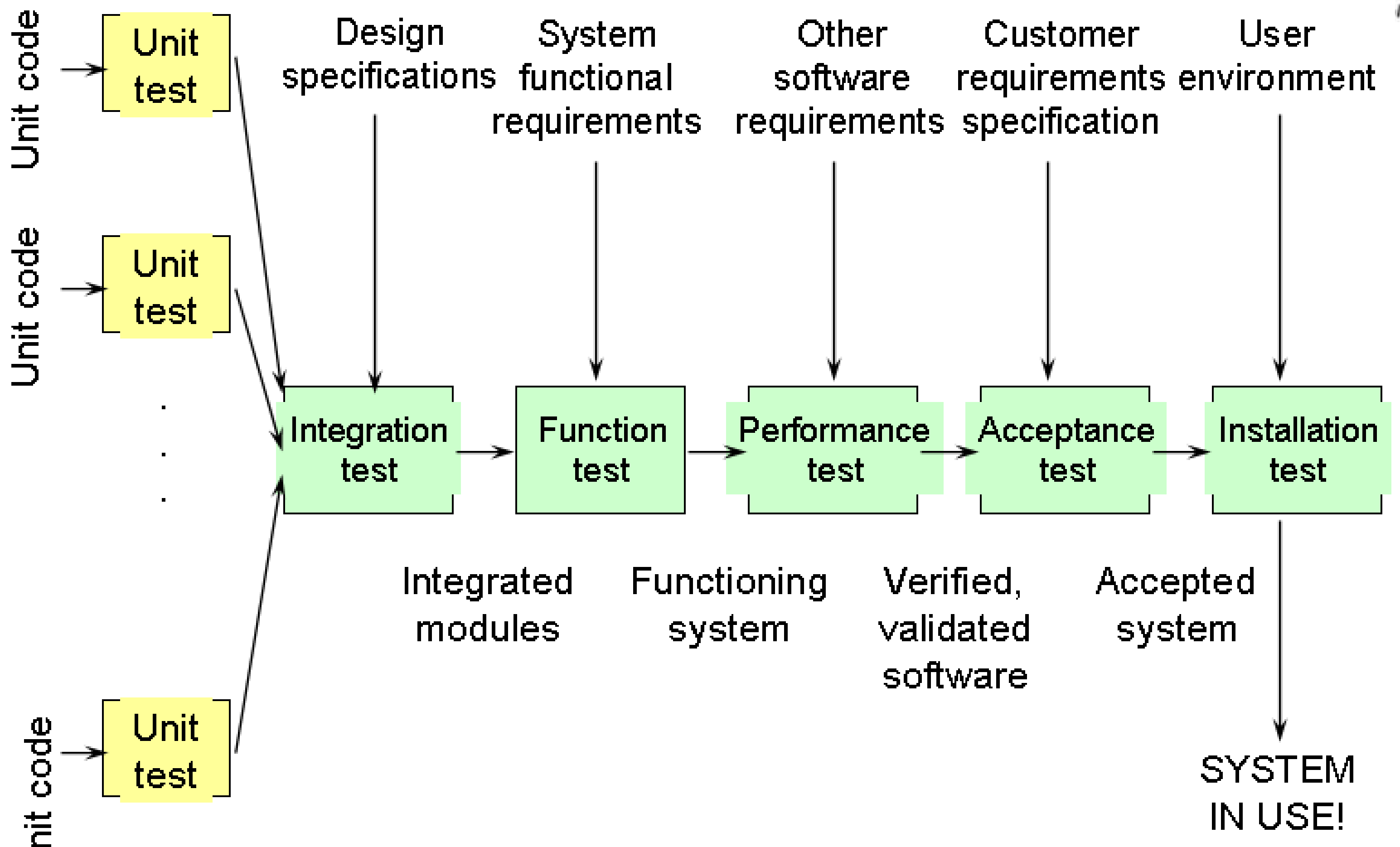
Verification, Validation, Testing

Verification	Validation	Testing
• Demonstration of consistency, completeness, and correctness of the software artifacts at each stage of and between each stage of the software life-cycle.	• The process of evaluating software at the end of the software development to ensure compliance with respect to the customer needs and requirements.	• Examination of the behavior of a program by executing the program on sample data • Testing is a verification technique used at the implementation stage.

Testing Levels or phases

Unit testing	• this is basically testing of a single function, procedure, class.
Integration testing	• this checks that units tested in isolation work properly when put together.
System testing	• here the emphasis is to ensure that the whole system can cope with real data, monitor system performance, test the system's error handling and recovery routines
Acceptance testing	• this checks if the overall system is functioning as required
Regression Testing	• this checks that the system preserves its functionality after maintenance and/or evolution tasks.

Testing Steps / What is to be tested?



[Pfleeger 2001]

Unit testing

Objectives

- To test the function of a program or unit of code such as a program or module
- To test internal logic
- To verify internal design
- To test path & conditions coverage
- To test exception conditions & error handling

When	After modules are coded
Input	Internal Application Design Master Test Plan & Unit Test Plan
Output	Unit Test Report
Who	Developer
Methods	White Box testing techniques & Test Coverage techniques
Tools	Debug, Re-structure, Code Analyzers & Path/statement coverage tools
Education	Testing Methodology & Effective use of tools

Integration testing

Objectives

- To technically verify proper interfacing between modules, and within sub-systems

When	After modules are unit tested
Input	Internal & External Application Design Master Test Plan & Integration Test Plan
Output	Integration Test report
Who	Developer
Methods	White and Black Box techniques Problem / Configuration Management
Tools	Debug, Re-structure & Code Analyzers
Education	Testing Methodology & Effective use of tools

System Testing

Objectives

- To verify that the system components perform control functions
- To perform inter-system test
- To demonstrate that the system performs both functionally and operationally as specified
- To perform appropriate types of tests relating to Transaction Flow, Installation, Reliability, Regression etc.

When	After Integration Testing
Input	Detailed Requirements & External Application Design Master Test Plan & System Test Plan
Output	System Test Report
Who	Development Team and Users
Methods	Problem / Configuration Management
Tools	Recommended set of tools
Education	Testing Methodology & Effective use of tools

Acceptance Testing

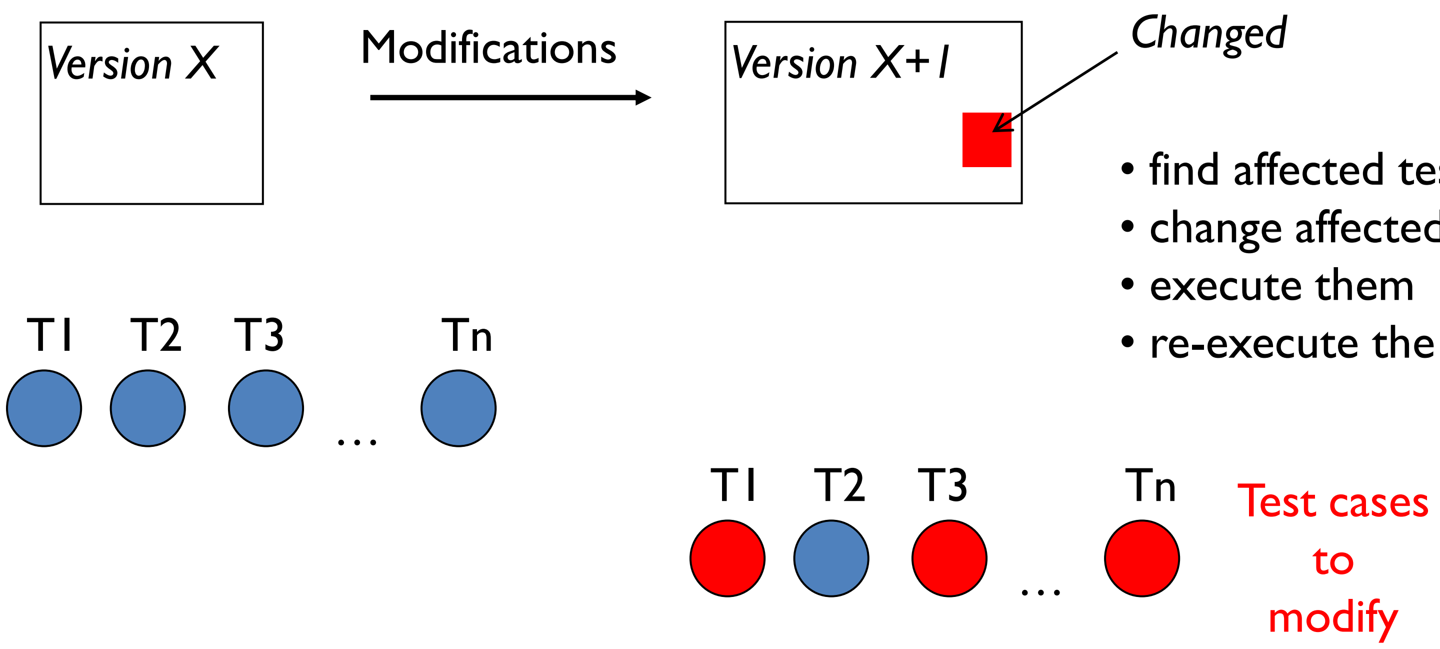
Objectives

- To verify that the system meets the user requirements

When	After System Testing
Input	Business Needs & Detailed Requirements Master Test Plan User Acceptance Test Plan
Output	User Acceptance Test report
Who	Users / End Users
Methods	Black Box techniques Problem / Configuration Management
Tools	Compare, keystroke capture & playback, regression testing
Education	Testing Methodology & Effective use of tools Product knowledge & Business Release Strategy

Regression testing

- Selective re-testing of a system or component to verify that modifications have not caused unintended effects



- find affected test cases (red)
- change affected test cases (red)
- execute them
- re-execute the others (ex. T2)

Software Testing Life cycle - Phases

- Requirements study
- Test Case Design and Development
- Test Execution
- Test Closure
- Test Process Analysis

Testing methodologies

- Black box testing
- White box testing
- Incremental testing
- Thread testing

Testing Levels/ Techniques	White Box	Black Box	Incremental	Thread
Unit Testing	X			
Integration Testing	X		X	X
System Testing		X		
Acceptance Testing		X		

Black-box test and white-box test

- **Black-box test (exterior test or functional test): Test of the input/output behavior without considering the inner structure of the test object (interface test)**
  - Incorrect or missing functions,
  - Interface errors,
  - Errors in data structures or external database access,
  - Performance errors, and
  - Initialization and termination errors.

- **White-box test (interior test or structure test): Test of the input/output behavior considering the inner structures (interface and structure test)**
  - The goal is to determine, for every possible path through the test object, the behavior of the test object in relation to the input data.

Comparison of Black and white box testing

White-box Testing	Black-box Testing
make up test cases based on how the data is known to be processed by the program	make up test cases based on the known requirements for input, output
requirements as designed in the detailed design doc or inspection of source code	requirements as given in the functional specifications or Business Requirements
tester needs explicit knowledge of internal workings of the item being tested	Tester does not need explicit knowledge of internal workings of the item being tested
more expensive	less expensive
requires source code	requires the executable
more laborious	less laborious

Bottom-up Testing

The advantages

- bottom-up testing prove to be the drawbacks of top-down testing (and vice versa).

- This method is solid and proven. The objects to be tested are known in full detail. It is often simpler to define relevant test cases and test data.

- This approach is psychologically more satisfying because the tester can be certain that the foundations for the test objects have been tested in full detail.

Software Testing - Categorization

- **Functional vs. Structural testing**
  - Functional testing is generating test cases based on the functionality of the software
  - Structural testing is generating test cases based on the structure of the program

- **Black box vs. White box testing**
  - Black box testing is same as functional testing. Program is treated as a black box, its internal structure is hidden from the testing process.
  - White box testing is same as structural testing. In white box testing internal structure of the program is taken into account

- **Module vs. Integration testing**
  - Module testing: Testing the modules of a program in isolation
  - Integration testing: Testing an integrated set of modules

Debugging

The two outcomes of debugging:

- Cause will be found, corrected, and removed
- Cause will not be found

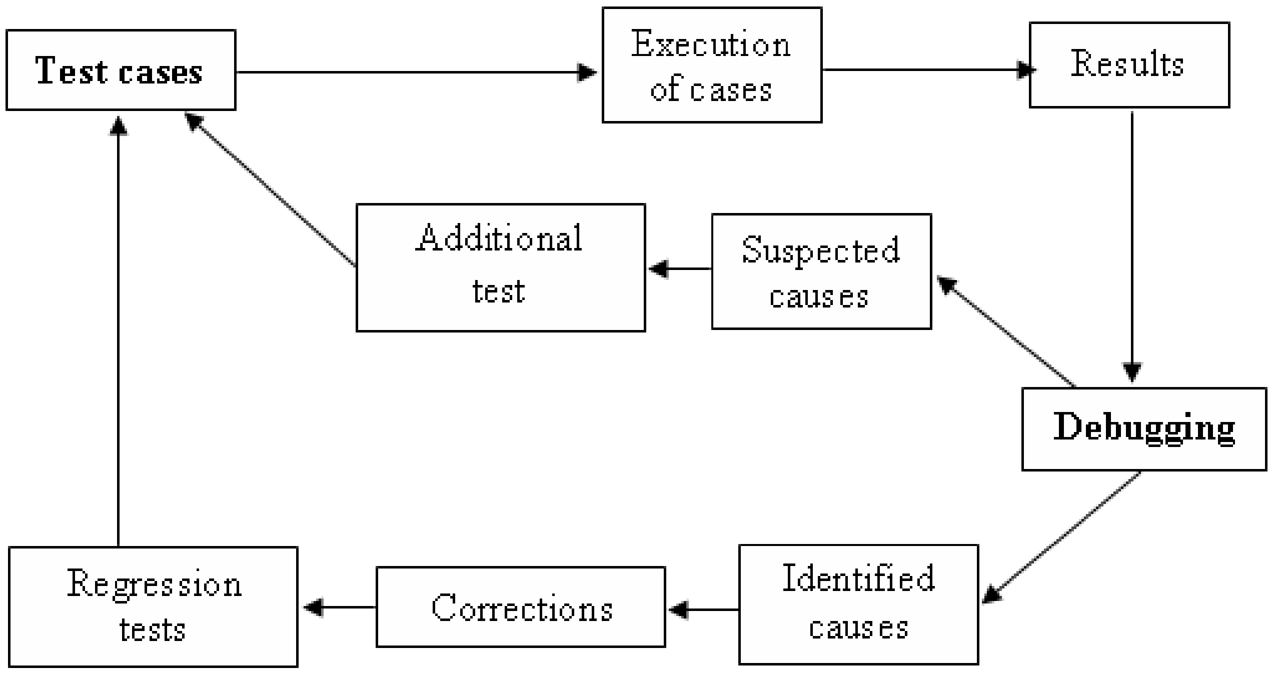


Figure: Debugging

Summary

In this module, we discussed:

- Verification, Validation and Testing
- Software testing categorization
- Unit testing, integration testing and System testing
- Testing requirements, design and specifications
- Testing steps

Quality Certifications & Standards

Module 6

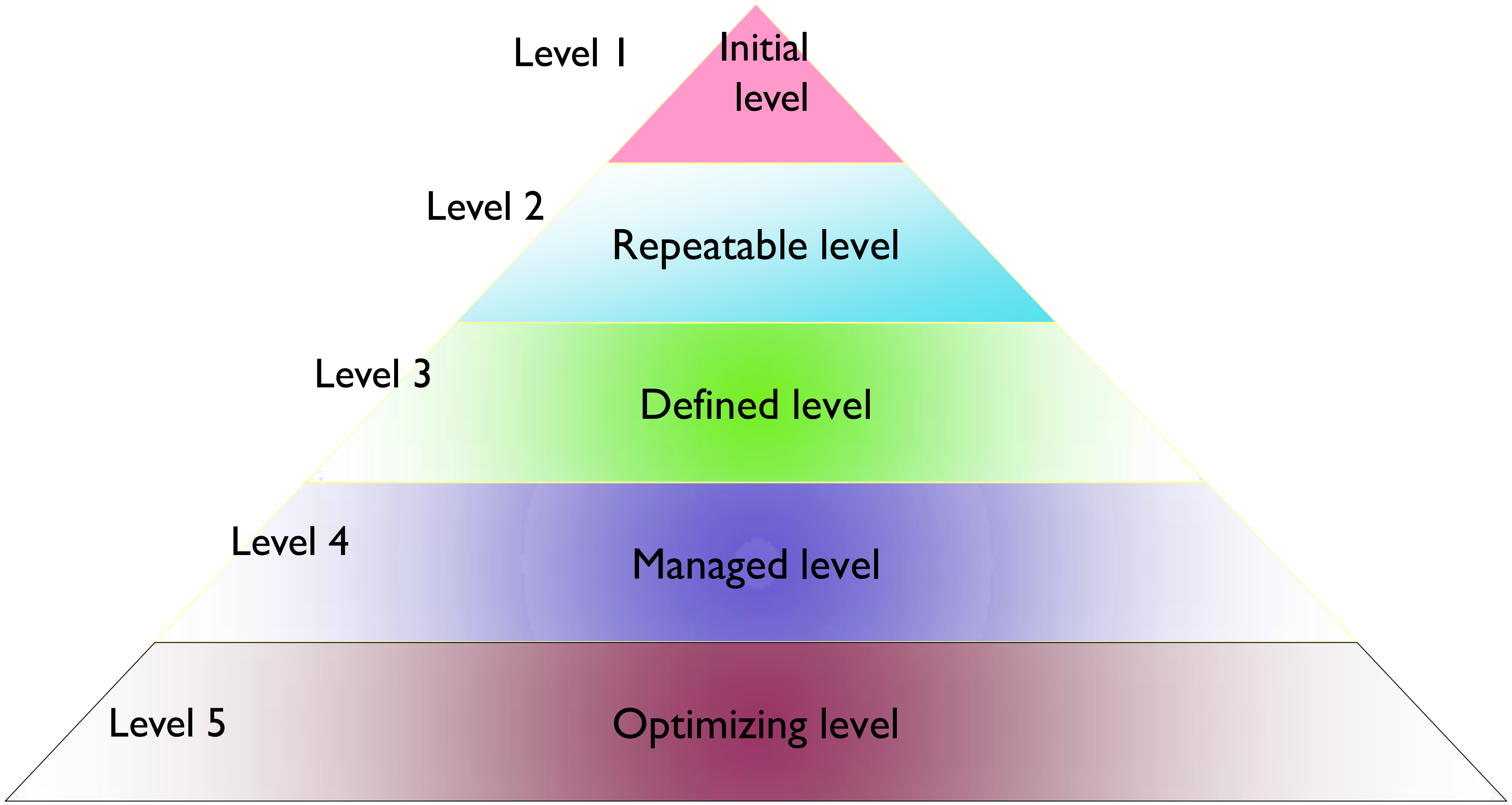
Objectives

At the end of this module, you will be able to:

- Identify various International Standards for Quality Assurances
- Recall all SDLC models, roles & responsibilities in each phase of project development lifecycle

Duration: 1 hr

CMM – Levels


Level 1
Initial level
Level 2
Repeatable level
Level 3
Defined level
Level 4
Managed level
Level 5
Optimizing level

Standards

- IEEE Standards
  - An entire set of standards for software is designed by Institute of Electrical and Electronics Engineers to be followed by testers
- ISO
  - International Organization for Standards
- Six Sigma
  - Zero Defect Orientation
- SPICE
  - Software Process Improvement and Capability Determination
- NIST
  - National Institute of Standards and Technology

Testing Standards

- External Standards
  - Familiarity with and adoption of industry test standards from organizations
- Internal Standards
  - Development and enforcement of the test standards that testers must meet

Summary

In this module, we discussed:

- Various International Standards for Quality Assurances
- A recap of all SDLC models, roles & responsibilities in each phase of project development lifecycle

Conclusion

Technical factors influences software maintenances are:

- Module independence
- Programming language
- Programming style
- Program validation
- Documentation

- ***Lehman's laws:***
  - **The law of continuing change**
  - **The law of increasing complexity**
  - **The law of large program evolution**
  - **The law of organizational stability**
  - **The law of conservation of familiarity**

What we call the beginning is often the end. And to make an end is to make a beginning. The end is where we start from.—T.S. Eliot

References

1. Frank T, Orlando K. *Essentials of Software Engineering*, Ed 2. New Delhi: Jones & Bartlett, 2010.
2. Srihari Techsoft. *Software Testing : An Overview.* Retrieved on Apr 29, 2011, from, http://www.careervarsity.com/careertools/SOFTWARETESTING.ppt
3. Software Engineering Institute Carnegie Mellon. *Software Engineering Body of Knowledge version 1.0, A.* Retrieved on Apr 29, 2011, from, http://www.sei.cmu.edu/library/abstracts/reports/99tr004.cfm
4. Krutchten, Philippe. *IBM Developer Works: The Rational Edge.* Retrieved on Apr 29, 2011, from, http://www.therationaledge.com/

Thank You