Software Engineering

Objectives

- To introduce software engineering and to explain its importance
- To set out the answers to key questions about software engineering
- To introduce ethical and professional issues and to explain why they are of concern to software engineers

Outlines Of the Lectures:

- Introduction
- Software Evolution
- Software Life Cycles
- Requirement Specification and Analysis
- Software Design Issues : DFD , UML
- Coding & Testing
- Project Planning, Risk Management and Estimation Technique
- Software Quality Management
- Discussions & Quiz/Examinations

Introduction

FAQs about software engineering

- What is software?
- What is software engineering?
- What is the difference between software engineering and computer science?
- What is a software process?
- What is a software process model?

FAQs about software engineering

- What are the costs of software engineering?
- What are software engineering methods?
- What is CASE (Computer-Aided Software Engineering)
- What are the attributes of good software?
- What are the key challenges facing software engineering?

What is software?

- Computer programs and associated documentation
- Software products may be developed for a particular customer or may be developed for a general market
- Software products may be
  - Generic - developed to be sold to a range of different customers
  - Custom - developed for a single customer according to their specification

What is Software

- Manuals and Operating Procedures
- Programs
- Design Documentation
- Models of Operating Environments
- Reliability Information
- Performance Evaluations

Characteristics of Softwares

- SW is developed or engineered, it is not manufactured in classical sense
- Software does not wear out
- Most SW is custom built, rather than being assembled from existing components
- Typically errors are high when software is built or changed and the error rates comes down
- The cost of correction / change increases exponentially when we move ahead in the life cycle of a SW project

What is Software Engineering

Software Engineering is the application of science and mathematics to make computers useful to people via software. What does Engineering means:

- ❑Builds real systems
- ❑Seeks to meet the needs of customers
- ❑Faces cost and schedule limits
- ❑Uses systematic methods
- ❑Exploits standard designs

What is Software Engineering - Contd.

Software Engineering is the *technological* and *managerial* discipline concerned with *systematic production* and *maintenance* of *software products* that are *developed* and *modified on time* and *within cost estimates*.

Software Engineering is needed most for large complex systems.

What is software engineering?

- Software engineering is an **engineering** discipline which is concerned with all aspects of software production
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available

Software Engineering

- Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, i.e. the application of engineering to software (IEEE)
- Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines (Software Engineering by Pressman)

What is the difference between software engineering and computer science?

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software
- Computer science theories are currently insufficient to act as a complete underpinning for software engineering

What is a software process?

- A set of activities whose goal is the development or evolution of software
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands

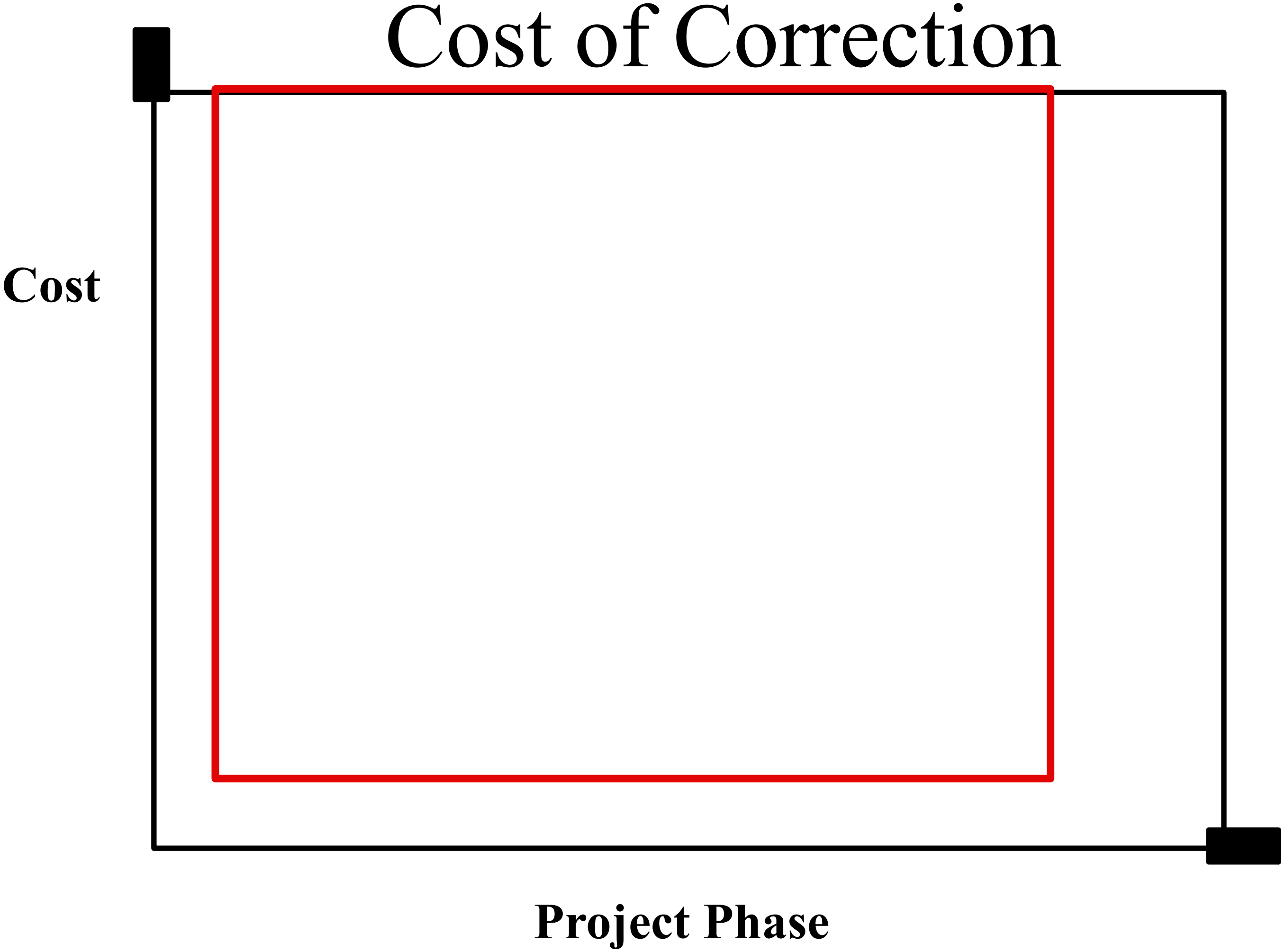
What is a software process model?

- A simplified representation of a software process, presented from a specific perspective
- Examples of process perspectives are
  - Workflow perspective - sequence of activities
  - Data-flow perspective - information flow
  - Role/action perspective - who does what
- Generic process models
  - Waterfall
  - Evolutionary development
  - Formal transformation
  - Integration from reusable components

Software Costs

- Software costs often dominate system costs. The costs of software on a PC are often greater than the hardware cost
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs
- Software engineering is concerned with cost-effective software development

Cost of Correction



What are the costs of software engineering?

- Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability
- Distribution of costs depends on the development model that is used

What are software engineering methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance
- Model descriptions
  - Descriptions of graphical models which should be produced
- Rules
  - Constraints applied to system models
- Recommendations
  - Advice on good design practice
- Process guidance
  - What activities to follow

What is CASE (Computer-Aided Software Engineering)

- Software systems which are intended to provide automated support for software process activities. CASE systems are often used for method support
- Upper-CASE
  - Tools to support the early process activities of requirements and design
- Lower-CASE
  - Tools to support later activities such as programming, debugging and testing

What are the attributes of good software?

- The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable
- Maintainability
  - Software must evolve to meet changing needs
- Dependability
  - Software must be trustworthy
- Efficiency
  - Software should not make wasteful use of system resources
- Usability
  - Software must be usable by the users for which it was designed

What are the key challenges facing software engineering?

- Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times
- Legacy systems
  - Old, valuable systems must be maintained and updated
- Heterogeneity
  - Systems are distributed and include a mix of hardware and software
- Delivery
  - There is increasing pressure for faster delivery of software

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals
- Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- *Confidentiality*
  - ❑Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- *Competence*
  - ❑Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outside their competence.

Issues of professional responsibility

- *Intellectual property rights*
  - ❑Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- *Computer misuse*
  - ❑Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

Outlines Of the Lectures:

- Introduction
- Software Evolution
- Software Life Cycles
- Requirement Specification and Analysis
- Software Design Issues : DFD , UML
- Coding & Testing
- Project Planning, Risk Management and Estimation Technique
- Software Quality Management
- Discussions & Quiz/Examinations

Software Evaluation

Software Evolution - Traditional View

Software evolution consists of activities required to keep a software system operational and responsive after it is accepted and placed into production.

Software Evolution - Present View

Software evolution includes management of all dynamic structures and activities involved in software development.

Evolution of Computer Systems

Vacuum-tube based

Transistor based

IC based

VLSI based

Non-Von Neuman Computing Systems

Evolution of Operating Systems

Batch processing

🟩

Time sharing

🟩

Multiprogramming

🟩

Multi-tasking

🟩

Multi-processing

Reasons for changes in Software Systems

✓Requirements for additional functionality, improved performance, or new configurations.

✓Modifications to existing requirements

✓Errors in the existing design or implementation

✓Excessive complexity in the design and implementation.

Tasks involved in changing a Software System

Understand the problem with the current version of the system.

Define the requirements for the change and the intended new behavior.

Locate the parts of the system documents affected by the change.

Design and implement the new version of the system.

Check all changes and test the new version of the system.

Importance of Software Evolution

Despite 50 years of progress, the software industry remains years - perhaps decades - short of the mature engineering discipline needed to meet the demands of an information-age society.

-Scientific American, September 1994 issue
"Software Chronic Crisis" by W. Gibbs

Areas of Concern in Software Development

- Self discipline is a badly-needed trait.
- Software problems became orders of magnitude larger.
- Range of applications widened greatly.
- The size of resulting programs is greatly inflated.
- Great willingness to make changes in specifications.
- Programmers are seldom restrained in changing programs.
- Can the software be “engineered” indeed?

-- R.W. Hamming in his Foreword in the Journal of Systms Integration, Volume 6, Numbers 1/2, March 1996.

Software Developers/Engineers

Programmer is an individual who is concerned with the details of ***implementing, packaging*** and ***modifying algorithms*** and ***data structures*** written in particular ***programming languages***.

Software Engineers are ***additionally*** concerned with the issues of ***analysis***, ***design***, ***verification***, ***documentation***, ***software maintenance*** and ***project management.***

What is the difference between software engineering and computer science?

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software
- Computer science theories are currently insufficient to act as a complete underpinning for software engineering

Evolution is not just another name for maintenance

Software evolution includes :

- **responses to requirements changes**
- **improvements to performance or clarity**
- **repairs of bugs**
- **and the overall organization of the development process**

Software evolution occurs throughout the life cycle

It includes :

specification-based development
incremental/phased development
requirements prototyping
version and configuration control
on-line documentation
testing code generation etc.

It captures dynamic aspects of software development

Issues in Software Evolution

Management issues - which parts of the system to be built

Configuration issues - tracking many versions of the system

Testing issues - which test cases could be influenced by a given change

Requirements issues - when the assumptions underlying a system specification are no longer valid.

Code restructuring issues

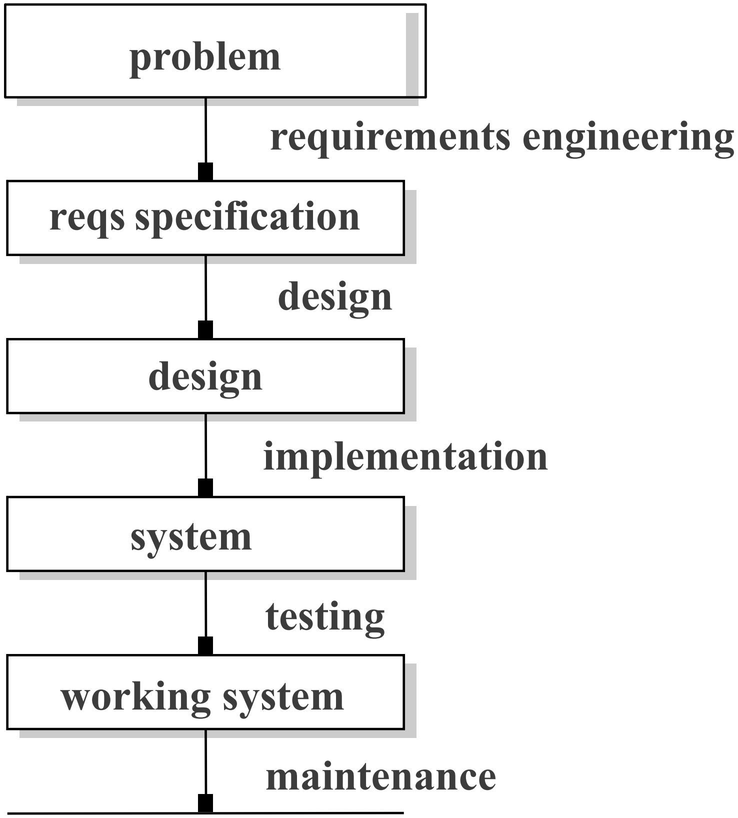
Performance improvement issues etc.

Outlines Of the Lectures:

- ❖ Introduction
- ❖ Software Evolution
- ❖ Software Life Cycles
- ❖ Requirement Specification and Analysis
- ❖ Software Design Issues : DFD , UML
- ❖ Coding & Testing
- ❖ Project Planning, Risk Management and Estimation Technique
- ❖ Software Quality Management
- ❖ Discussions & Quiz/Examinations

Software Life Cycle

Simple life cycle model

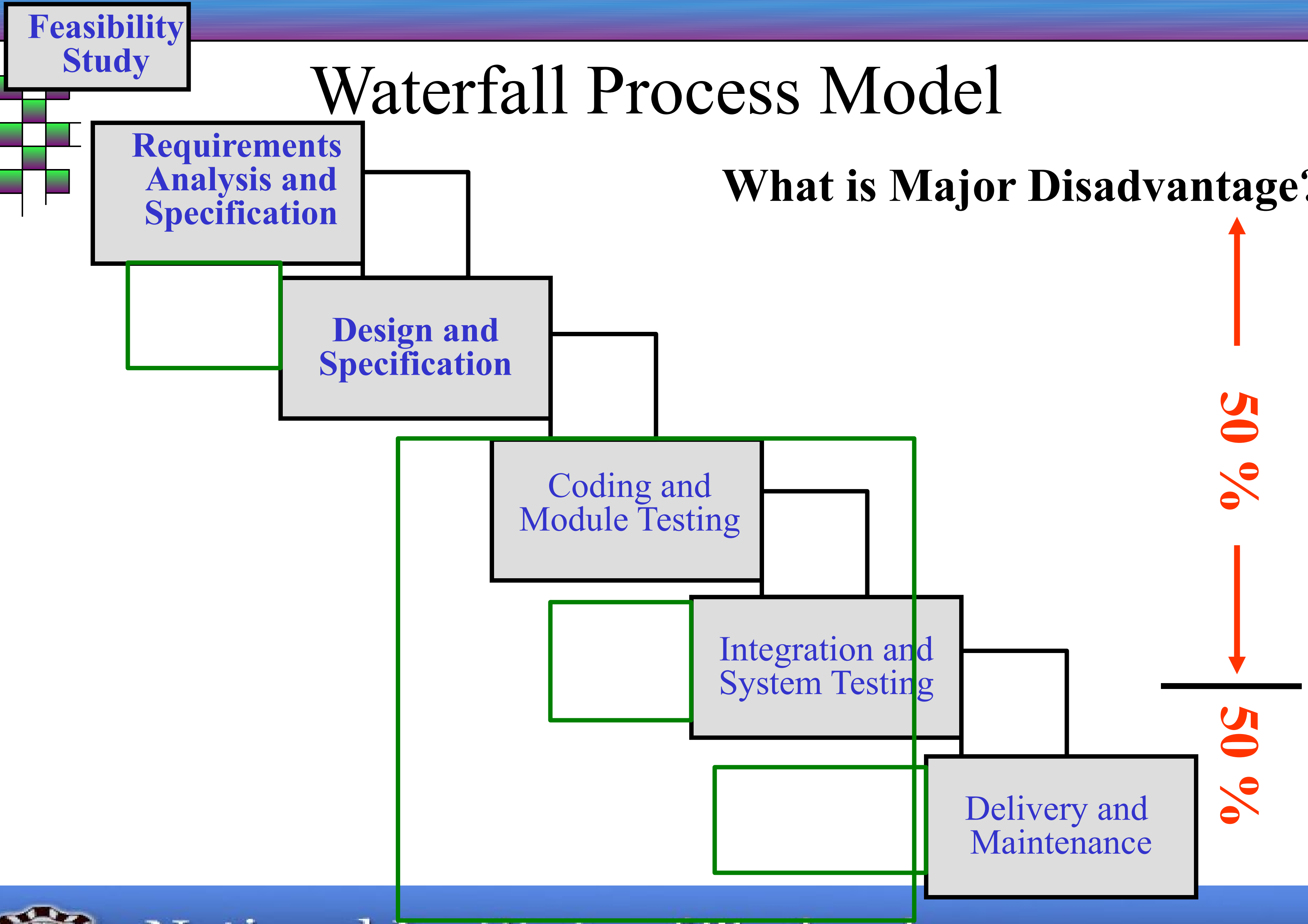


SDLC Model

A framework that describes the activities performed at each stage of a software development project.

Waterfall Process Model

What is Major Disadvantage?


Feasibility Study
Requirements Analysis and Specification
Design and Specification
Coding and Module Testing
Integration and System Testing
Delivery and Maintenance
50 %
50 %

Software Lifecycle of Waterfall Model

- Requirements Analysis and Specification
  - ❑ What is the Problem to Solve?
  - ❑ What Does Customer Need/Want?
  - ❑ Interactions Between SE and Customer
  - ❑ Identify and Document System Requirements
  - ❑ Generate User Manuals and Test Plans
- Design and Specification
  - ❑ How is the Problem to be Solved?
  - ❑ High-Level Design
  - ❑ Determine Components/Modules
  - ❑ Transition to Detailed Design
  - ❑ Detail Functionality of Components/Modules

Software Lifecycle of Waterfall Model

- Coding and Module Testing
  - Writing Code to Meet Component/Module Design Specifications
  - Individual Test Modules in Isolation
  - Drivers and Stubs to Simulate Behavior
- Integration and System Testing
  - Integration of Components/Modules into Subsystems
  - Integration of Subsystems into Final Program
- Delivery and Maintenance
  - System Delivered to Customer/Market
  - Bug Fixes and Version Releases Over Time

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

Waterfall Model - Contd.

Concerns :Software development may proceed linearly from analysis down to coding.

- The results of each phase are frozen before proceeding to the next phase.

Consequence - requirements and design specifications may be frozen at an early stage of development.

The entire planning is oriented to a single delivery date.

Waterfall Model - Contd.

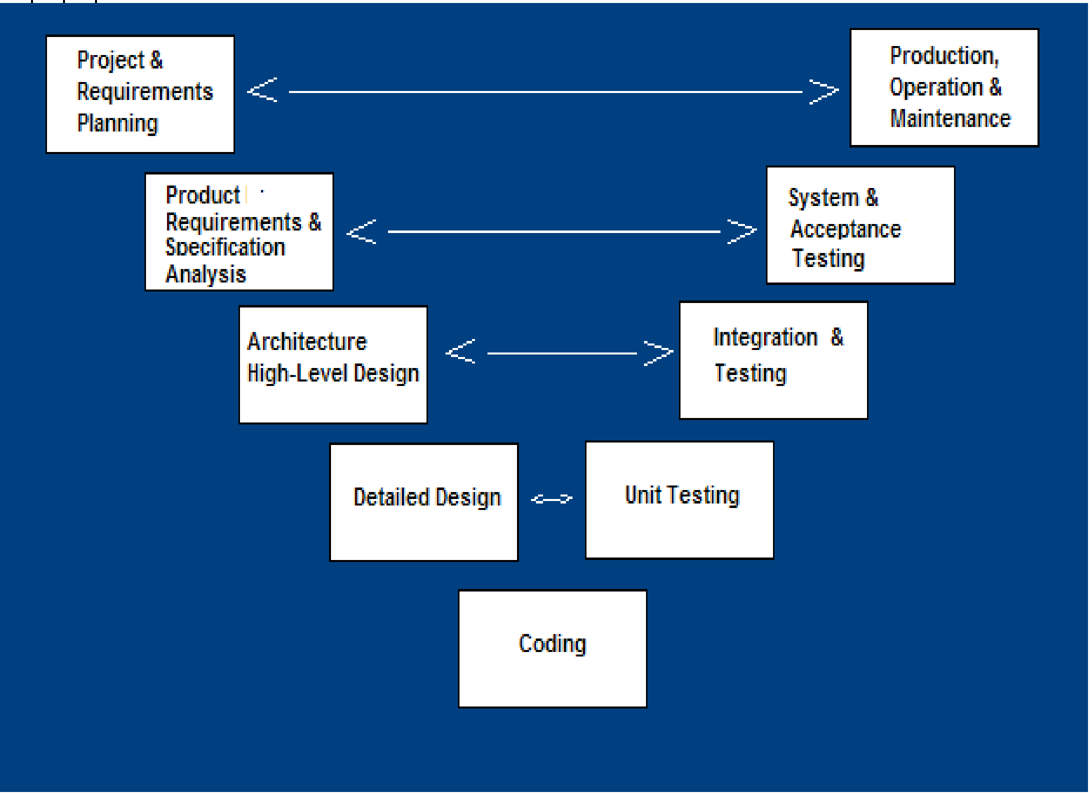
Concerns :

- Difficulty in accurate estimation of resources with only limited information.
- The user often does not know the exact requirements.

✓ The waterfall model does not stress the need for anticipating changes

✓ The model enforces standards that are heavily based on the production of certain documents at certain specific times.

V-Shaped SDLC Model


Project & Requirements Planning
Production, Operation & Maintenance
Product Requirements & Specification Analysis
System & Acceptance Testing
Architecture High-Level Design
Integration & Testing
Detailed Design
Unit Testing
Coding

- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component

- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected
- Coding – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

Lightweight (agile) approaches

- Prototyping
- Incremental development
- RAD, DSDM
- XP

The Agile Manifesto

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Prototyping

Requirements elicitation is difficult

- ❑software is developed because the present situation is unsatisfactory
- ❑however, the desirable new situation is as yet unknown

Prototyping is used to obtain the requirements of some aspects of the system

Prototyping should be a relatively cheap process

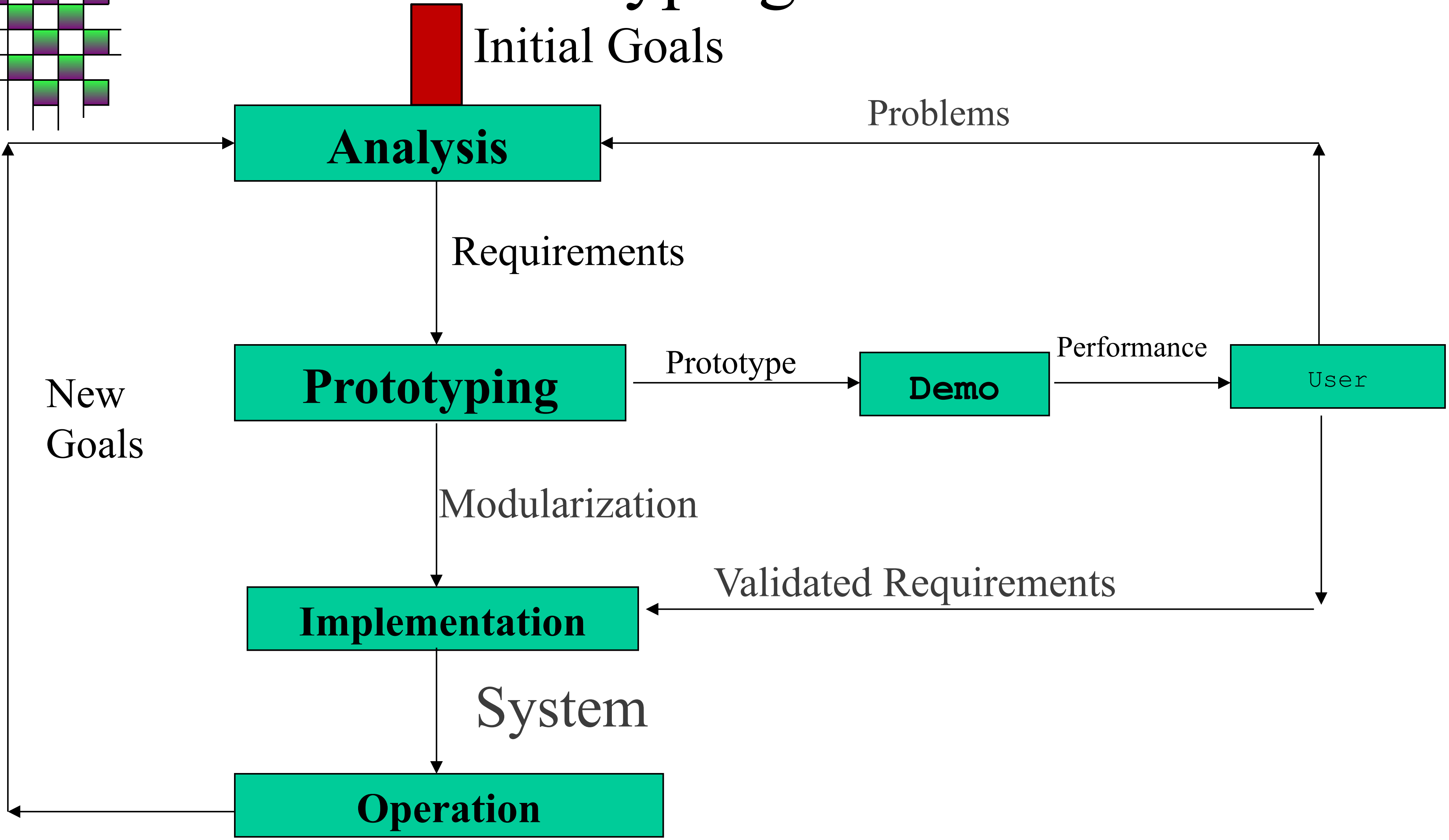
- ❑use rapid prototyping languages and tools
- ❑not all functionality needs to be implemented
- ❑production quality is not required

Prototyping

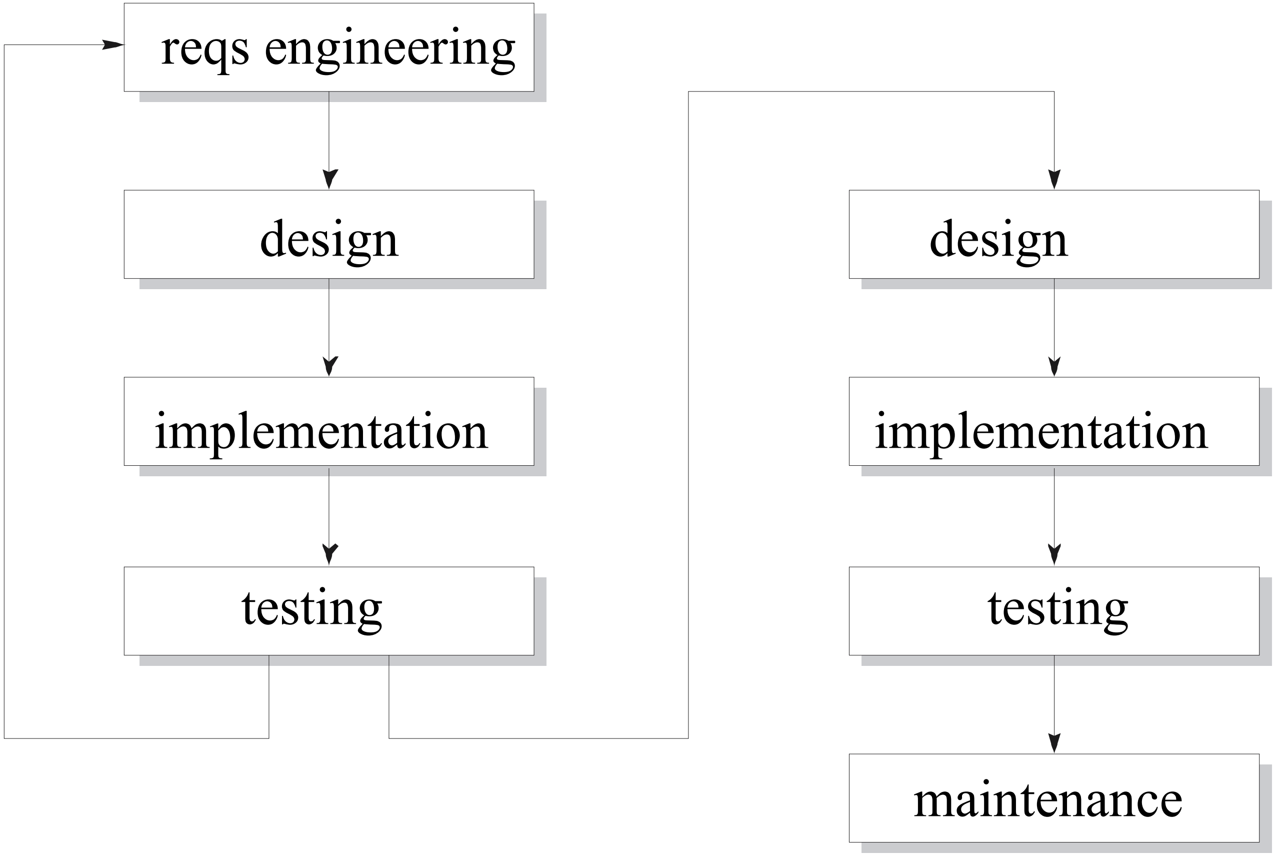
- Prototypes provide inexpensive demonstration of the essential aspects of the proposed new system behavior. This approach is based on the realistic assumption that many iterative changes to proposed system behavior are necessary to formulate an acceptable version of requirements.

- Prototypes are constructed prior to production version to -Gain information that guides analysis and design, and Support generation of the production version.

Prototyping - Contd.



Prototyping as a tool for requirements engineering



Prototyping

- *throwaway prototyping*: the n-th prototype is followed by a waterfall-like process (as depicted on previous slide)

- *evolutionary prototyping*: the nth prototype is delivered

Prototyping, advantages

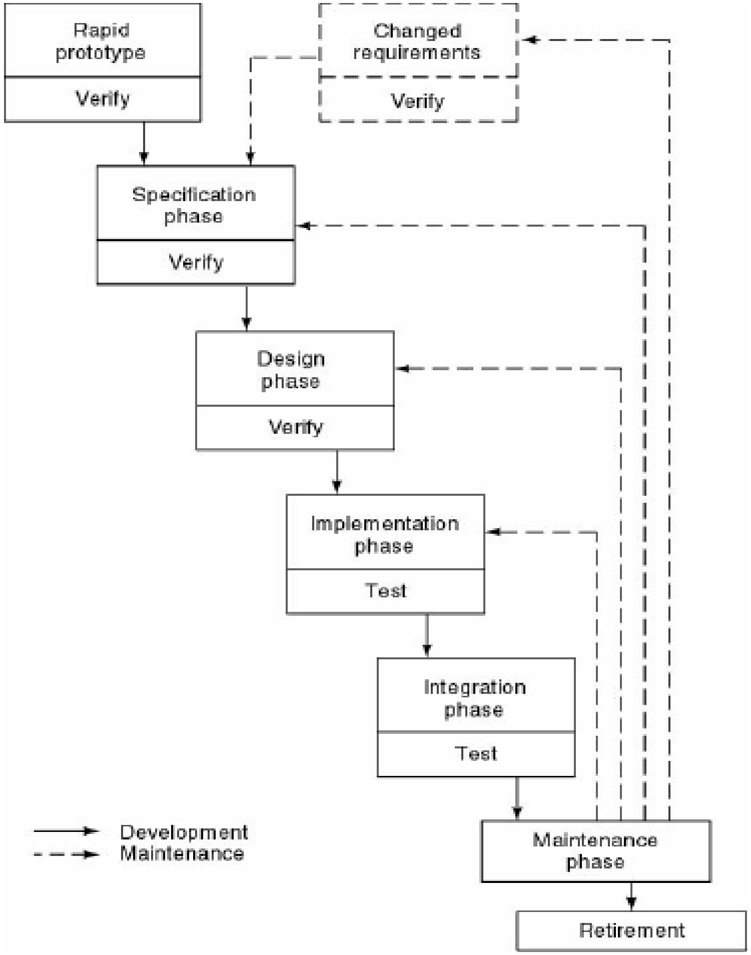
- The resulting system is easier to use
- User needs are better accommodated
- The resulting system has fewer features
- Problems are detected earlier
- The design is of higher quality
- The resulting system is easier to maintain
- The development incurs less effort

Prototyping, disadvantages

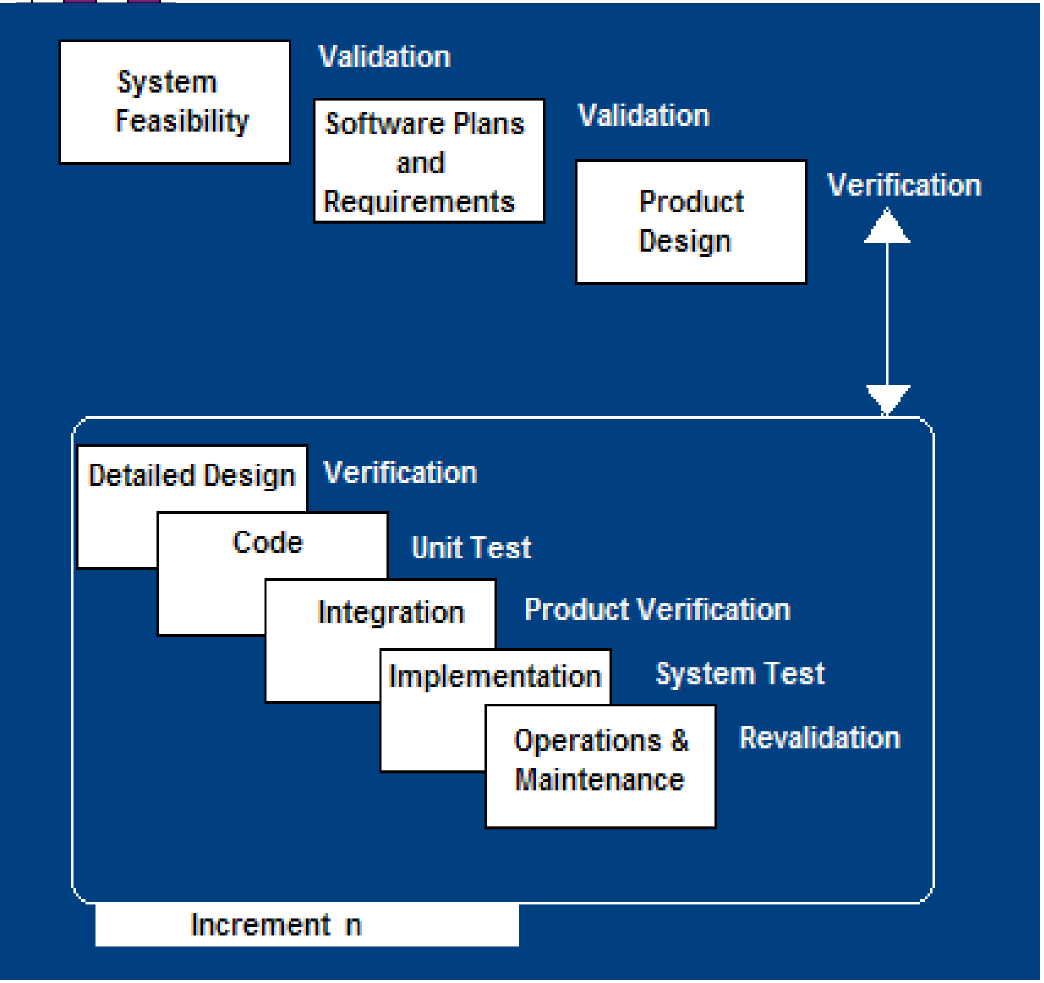
- The resulting system has more features
- The performance of the resulting system is worse
- The design is of less quality
- The resulting system is harder to maintain
- The prototyping approach requires more experienced team members

Waterfall and Rapid Prototyping Models

- Waterfall model
  - Many successes
  - Client needs
- Rapid prototyping model
  - Cannot be used for robust applications
- Solution
  - Rapid prototyping for requirements phase
  - Waterfall for rest of life cycle



Incremental SDLC Model



- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses “divide and conquer” breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

Incremental Model Weaknesses

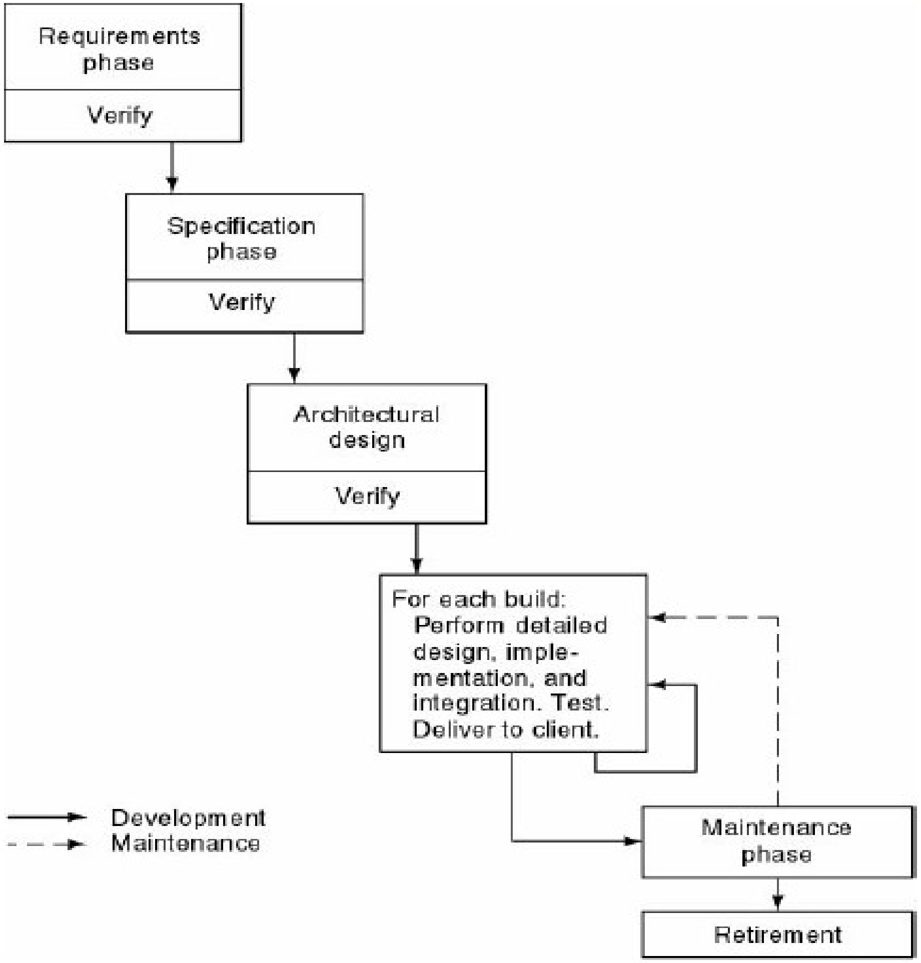
- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

When to use the Incremental Model

- Risk, funding, schedule, program complexity, or need for early realization of benefits.
- Most of the requirements are known up-front but are expected to evolve over time
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules
- On a project with new technology

Incremental Model

- Divide project into *builds*

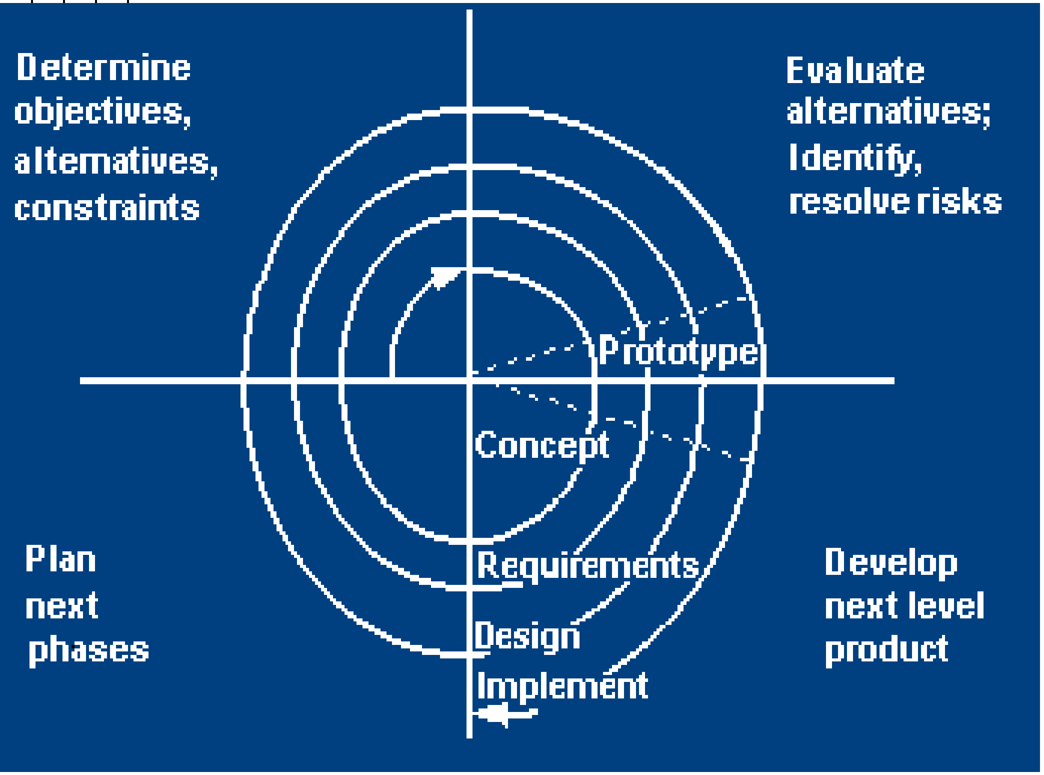


Spiral Model

This model provides a view of the production process that supports risk management.

It may be considered as a meta-model - since it can accommodate any process development model.

Spiral SDLC Model


Determine objectives, altematives, constraints
Evaluate alternatives; Identify, resolve risks
Prototype
Concept
Requirements
Design
Implement
Plan next phases
Develop next level product

- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model

Spiral Quadrant-1
Determine objectives, alternatives and constraints

- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, interface, etc.

Spiral Quadrant-2
Evaluate alternatives, identify and resolve risks

- Study alternatives relative to objectives and constraints
- Identify risks (lack of experience, new technology, tight schedules, poor process, etc.
- Resolve risks (evaluate if money could be lost by continuing system development

Spiral Model – Quadrant Details.

Stage-1 identifies the objectives of the portion of the product under consideration - in terms of qualities to achieve. It also identifies alternatives - whether to buy, design or reuse any software - and the constraints on the application of the alternatives.

Stage-2 evaluates the alternatives - identifies the potential risk areas and deals with them. Risk assessment may require prototyping and/or simulation.

Spiral Quadrant-3
Develop next-level product

- Typical activities:
  - Create a design
  - Review design
  - Develop code
  - Inspect code
  - Test product

Spiral Quadrant-4
Plan next phase

- Typical activities
  - Develop project plan
  - Develop configuration management plan
  - Develop a test plan
  - Develop an installation plan

Spiral Model – Quadrant Details.

Stage-3 consists of developing and verifying the next level in product development process. In less understood end-user applications, this step may again be evolutionary in nature.

Stage-4 consists of reviewing the results of the stages traversed so far and planning for the next iteration of the spiral, if any.

If risks cannot be resolved, project is immediately terminated

Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

Spiral Model Weaknesses

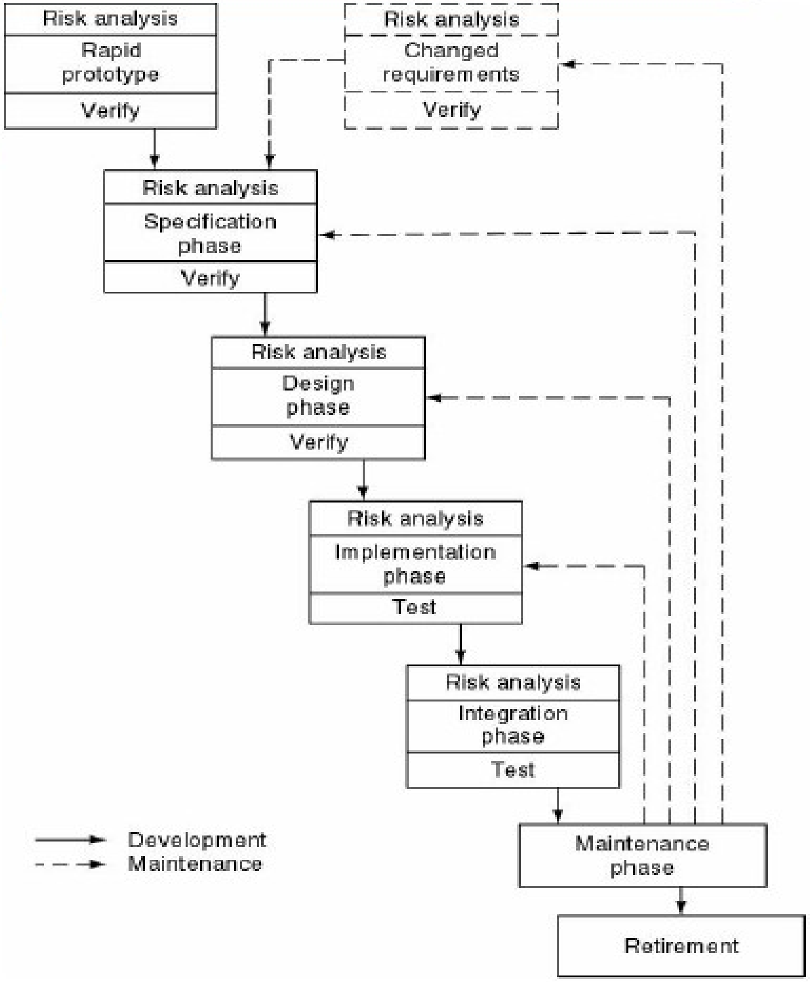
- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

When to use Spiral Model

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

Spiral Model

- Simplified form
  - Waterfall model plus risk analysis
- Precede each phase by
  - Alternatives
  - Risk analysis
- Follow each phase by
  - Evaluation
  - Planning of next phase



Spiral Model - Contd.

Risks are potentially adverse circumstances that may impair the development process and the quality of products.

Risk management is a discipline whose objectives are to identify, address and eliminate software risk items before they become either threats to successful software operation or a major source of expensive software rework.

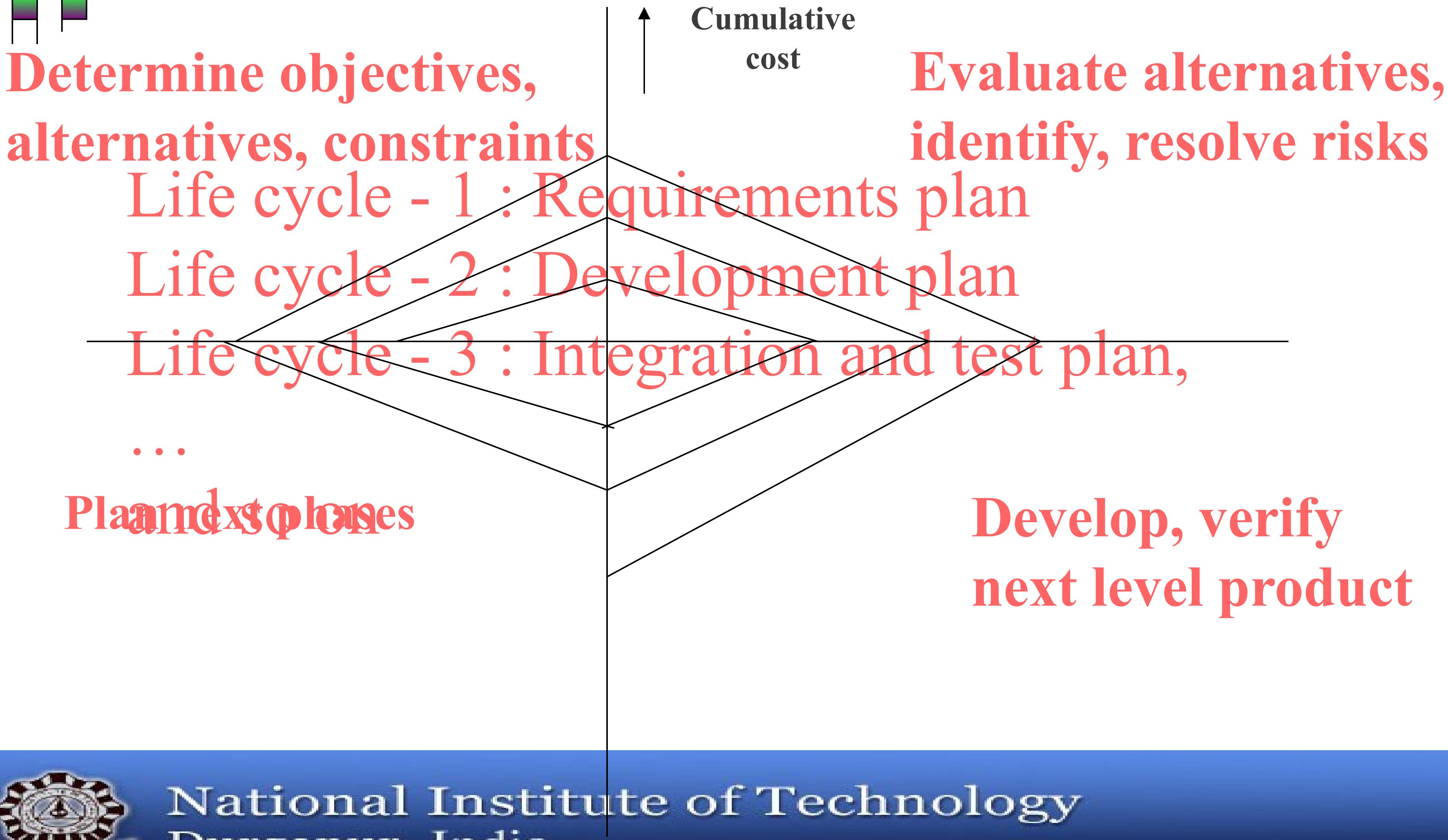
Spiral Model - Contd.

The spiral model is cyclic in nature.

Each cycle of the spiral consists of four stages - and each stage is represented by one quadrant of the Cartesian diagram.

The radius of the spiral represents the cost accumulated so far in the process and the angular dimension represents the progress in the process.

Spiral Model - Contd.



Determine objectives, alternatives, constraints

Evaluate alternatives, identify, resolve risks

Life cycle - 1 : Requirements plan

Life cycle - 2 : Development plan

Life cycle - 3 : Integration and test plan,

… and so on

Plan next phases

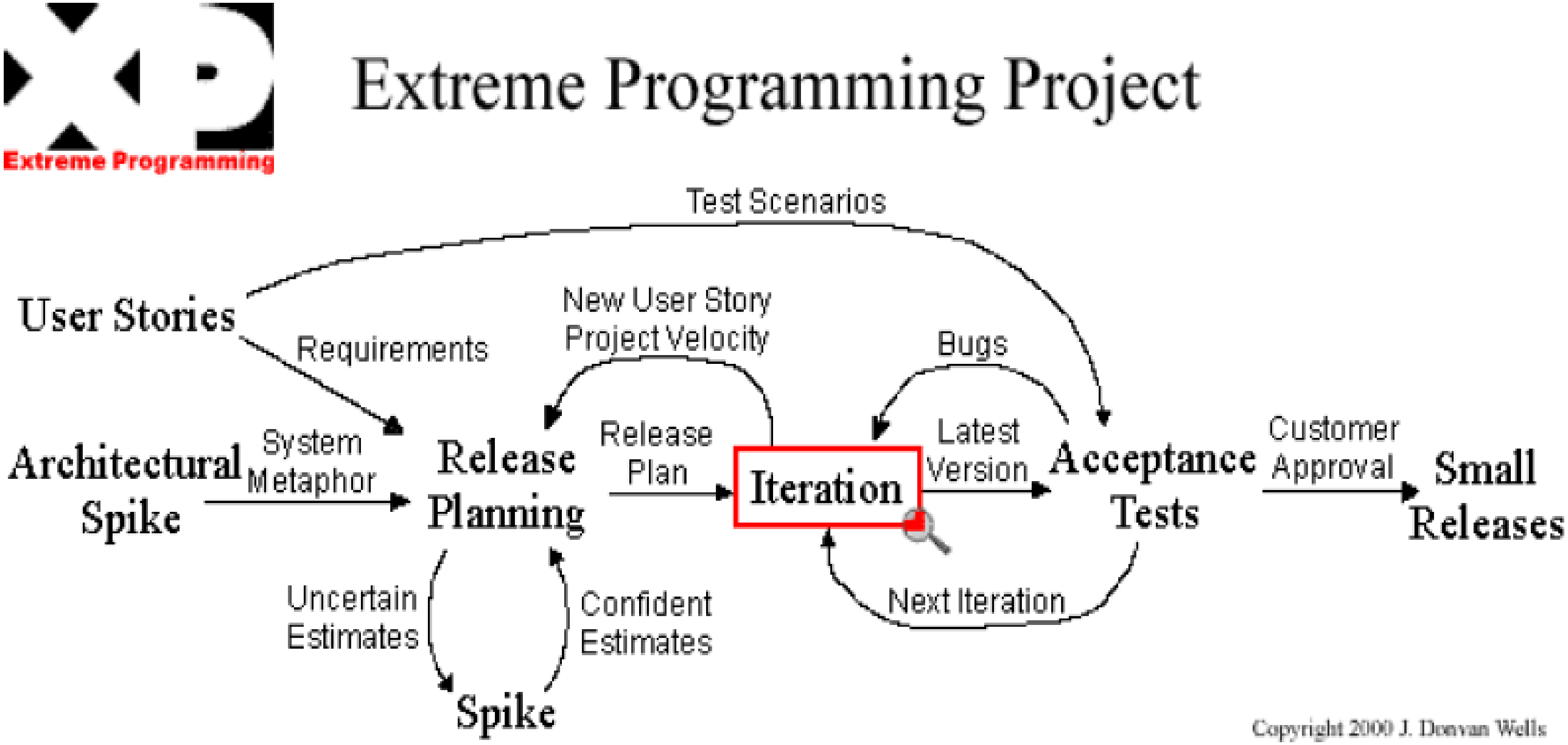
Develop, verify next level product

Spiral Model - Contd.

The spiral model emphasizes the issue of robustness together with correctness.

After one cycle of the spiral, unstated requirements are checked as part of the robustness of the application - and they become part of the specification in the next cycle.

Extreme Programming



Extreme Programming - XP

For small-to-medium-sized teams developing software with vague or rapidly changing requirements

Coding is the key activity throughout a software project

- Communication among teammates is done with code
- Life cycle and behavior of complex objects defined in test cases – again in code

XP Practices

1. Planning game – determine scope of the next release by combining business priorities and technical estimates
2. Small releases – put a simple system into production, then release new versions in very short cycle
3. Metaphor – all development is guided by a simple shared story of how the whole system works
4. Simple design – system is designed as simply as possible (extra complexity removed as soon as found)
5. Testing – programmers continuously write unit tests; customers write tests for features
6. Refactoring – programmers continuously restructure the system without changing its behavior to remove duplication and simplify

XP Practices

1. Pair-programming -- all production code is written with two programmers at one machine
2. Collective ownership – anyone can change any code anywhere in the system at any time.
3. Continuous integration – integrate and build the system many times a day – every time a task is completed.
4. 40-hour week – work no more than 40 hours a week as a rule
5. On-site customer – a user is on the team and available full-time to answer questions
6. Coding standards – programmers write all code in accordance with rules emphasizing communication through the code

XP is “extreme” because

Commonsense practices taken to extreme levels

- If code reviews are good, review code all the time (pair programming)
- If testing is good, everybody will test all the time
- If simplicity is good, keep the system in the simplest design that supports its current functionality. (simplest thing that works)
- If design is good, everybody will design daily (refactoring)
- If architecture is important, everybody will work at defining and refining the architecture (metaphor)
- If integration testing is important, build and integrate test several times a day (continuous integration)
- If short iterations are good, make iterations really, really short (hours rather than weeks)

Extreme Programming

- Somewhat controversial new approach
- Stories (features client wants)
- Estimate duration and cost of each story
- Select stories for next build
- Each build is divided into tasks
- Test cases for task are drawn up first
- Pair programming
- Continuous integration of tasks

Unusual Features of XP

- Computers are put in center of large room lined with cubicles
- Pair programming with shared computers
- Client representative is always present
- Cannot work overtime for 2 successive weeks
- No specialization
- Refactoring

Evaluating XP

- XP has had some successes
- Good when requirements are vague or changing
- Too soon to evaluate XP

Object-Oriented Life-Cycle Models

- Need for iteration within and between phases
  - Fountain model
  - Recursive/parallel life cycle
  - Round-trip gestalt
  - Unified software development process
- All incorporate some form of
  - Iteration
  - Parallelism
  - Incremental development
- Danger
  - CABTAB

Code a bit, test a bit – CABTAB

- Haphazard undisciplined approach

Software Development Process Models - Conclusion

It is impossible to reconcile all the conflicting requirements of the software development process in a new universal blanket process model.

The first step of any software process should always consist of selecting or designing the model to be followed in the succeeding steps.

Evolutionary model, in some form or other, is likely to be the mostly used model at least in near future.

SDLC brief

1. Classical water fall model
2. Iterative waterfall model
3. Prototype Model: Rapid prototype model
4. Evolutionary model: Incremental model
5. Meta model: Spiral model

Conclusions

- Different life-cycle models
- Each with own strengths
- Each with own weaknesses
- Criteria for deciding on a model include
  - The organization
  - Its management
  - Skills of the employees
  - The nature of the product
- Best suggestion
  - “Mix-and-match” life-cycle model

Life Cycle Selection

1. **Characteristics of the software:** If simple data processing, go for iterative waterfall model; object oriented development, go for evolutionary model.
2. **Characteristics of the development team:** If team is experienced, go for iterative waterfall model. If development team is novice, go for prototype model.
3. **Characteristics of the customer:** If customer is well competent in computer

The Four P's of Software Engineering

- Project – the task at hand
- People – by whom it is done
- Process – the manner it is done
- Product – the artifacts produced

Outlines Of the Lectures:

- Introduction
- Software Evolution
- Software Life Cycles
- Requirement Specification and Analysis
- Software Design Issues : DFD , UML
- Coding & Testing
- Project Planning, Risk Management and Estimation Technique
- Software Quality Management
- Discussions & Quiz Examinations

Requirement Specification & Analysis

Phase	Documents	QA
Requirement Definition	• Rapid prototype, or • Requirements document	• Rapid prototype • Reviews
Functional Specification	• Specification document (specifications) • Software Product Management Plan	• Traceability • FS Review • Check the SPMP
Design	• Architectural Design • Detailed Design	• Traceability • Review
Coding	• Source code • Test cases	• Traceability • Review • Testing
Integration	• Source code • Test cases	• Integration testing • Acceptance testing
Maintenance	• Change record • Regression test cases	• Regression testing

Jackson’s System Development

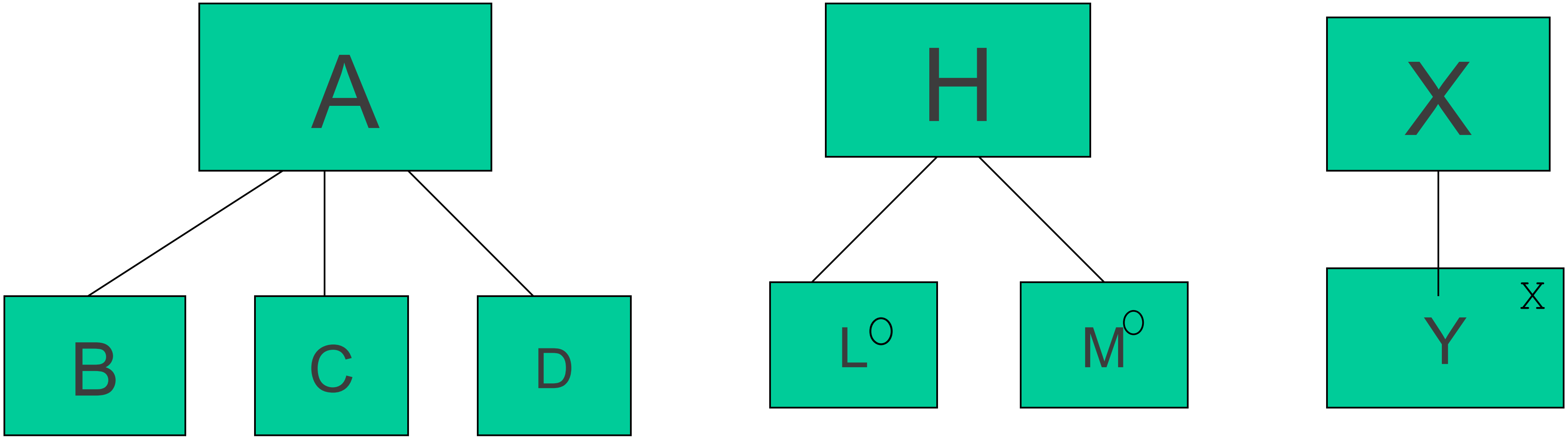
JSD suggests a technique that represents a mixture of descriptive approaches based on object-oriented design and functional decomposition.

JSD addresses all aspects of software development - from analysis to implementation.

In JSD, software development proceeds through a sequence of three stages - the modeling stage, the network stage, and the implementation stage.

Jackson’s System Development - Contd.

Jackson’s System Development - Contd.



Sequence

Selection

Iteration

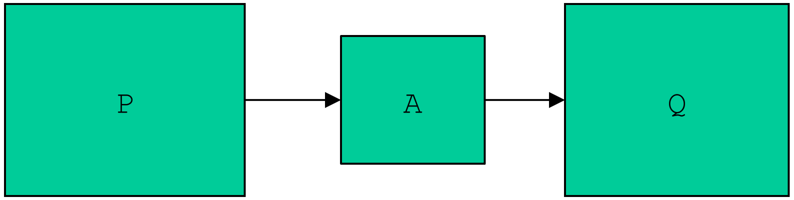
A : FORM B : GET
C : FILL_IN D : CHECK_IN

O stands for selection

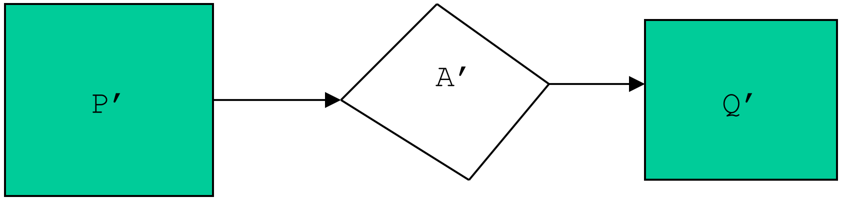
X stands for iteration

Jackson’s System Development - Contd.

In the network stage, the entire system is a network of interconnected and communicating processes - described by a system specification network (SSN)



Connection by data stream



Connection by state vector