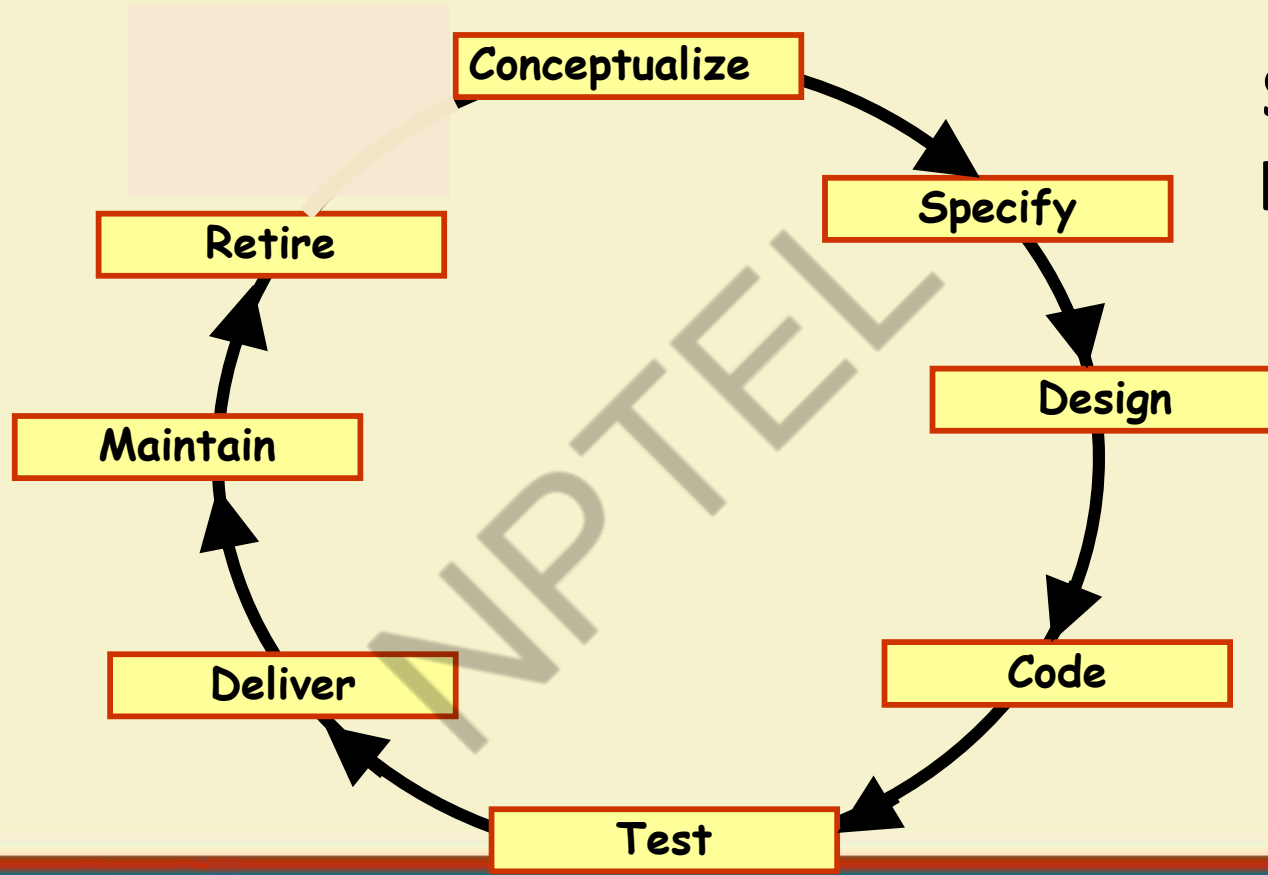


Life Cycle Models

Software Life Cycle



Life Cycle Model

- A software life cycle model (also process model or SDLC):
 - A descriptive and diagrammatic model of software life cycle:
 - Identifies all the activities undertaken during product development,
 - Establishes a precedence ordering among the different activities,
 - Divides life cycle into phases.

Life Cycle Model (CONT.)

- Each life cycle phase consists of several activities.
 - For example, the design stage might consist of:
 - structured analysis
 - structured design
 - Design review

Why Model Life Cycle?

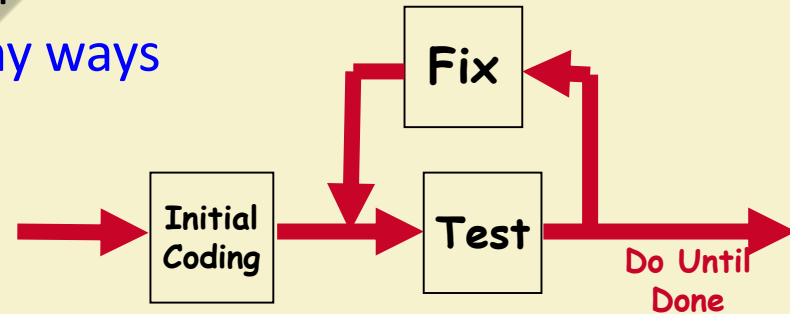
- A graphical and written description:
 - **Helps common understanding of activities among the software developers.**
 - **Helps to identify inconsistencies, redundancies, and omissions in the development process.**
 - **Helps in tailoring a process model for specific projects.**

Life Cycle Model (CONT.)

- The development team must identify a suitable life cycle model:
 - and then adhere to it.
 - Primary advantage of adhering to a life cycle model:
 - Helps development of software in a systematic and disciplined manner.

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
 - The problem is within the grasp of an individual.
 - He has the freedom to decide his exact steps and still succeed --- called Exploratory model--- One can use it in many ways
 - Code → Test → Design
 - Code → Design → Test → Change Code →
 - Specify → code → Design → Test → etc.



Life Cycle Model (CONT.)

- When software is being developed by a team:
 - There must be a precise understanding among team members as to when to do what,
 - Otherwise, it would lead to chaos and project failure.

Life Cycle Model (CONT.)

- A software project will never succeed if:
 - one engineer starts writing code,
 - another concentrates on writing the test document first,
 - yet another engineer first defines the file structure
 - another defines the I/O for his portion first.

Phase Entry and Exit Criteria

- A life cycle model:



- defines entry and exit criteria for every phase.
- A phase is considered to be complete:
 - only when all its exit criteria are satisfied.

Life Cycle Model (CONT.)

- What is the phase exit criteria for the software requirements specification phase?
 - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.
- A phase can start:
 - Only if its phase-entry criteria have been satisfied.

Life Cycle Model: Milestones

- Milestones help software project managers:
 - Track the progress of the project.
 - Phase entry and exit are important milestones.



Life Cycle and Project Management

- When a life cycle model is followed:
 - The project manager can at any time fairly accurately tell,
 - At which stage (e.g., design, code, test, etc.) the project is.

Project Management Without Life Cycle Model

- It becomes very difficult to track the progress of the project.
 - The project manager would have to depend on the guesses of the team members.
- This usually leads to a problem:
 - known as the **99% complete syndrome**.

Project Deliverables: Myth and Reality

Myth:

The only deliverable for a successful project is the working program.

Reality:

Documentation of **all aspects** of software development are needed to help in operation and maintenance.

- Many life cycle models have been proposed.
- We confine our attention to only a few commonly used models.

- Waterfall
- V model,
- Evolutionary,
- Prototyping
- Spiral model,
- Agile models

Traditional models

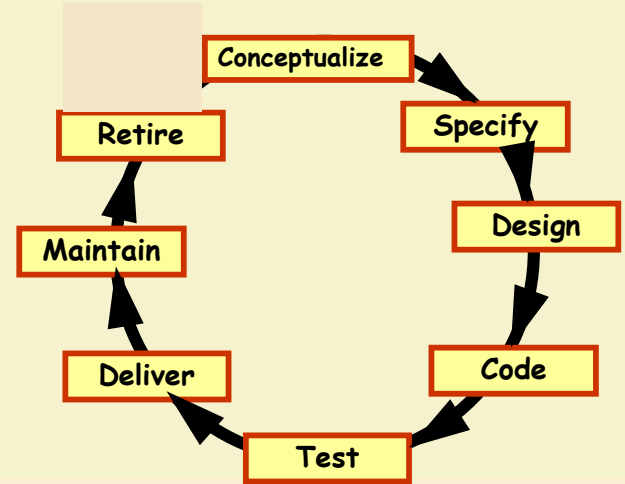
Life Cycle Model (CONT.)

- Software life cycle (or software process):
 - Series of identifiable stages that a software product undergoes during its life time:
 - Feasibility study
 - Requirements analysis and specification,
 - Design,
 - Coding,
 - Testing
 - Maintenance.

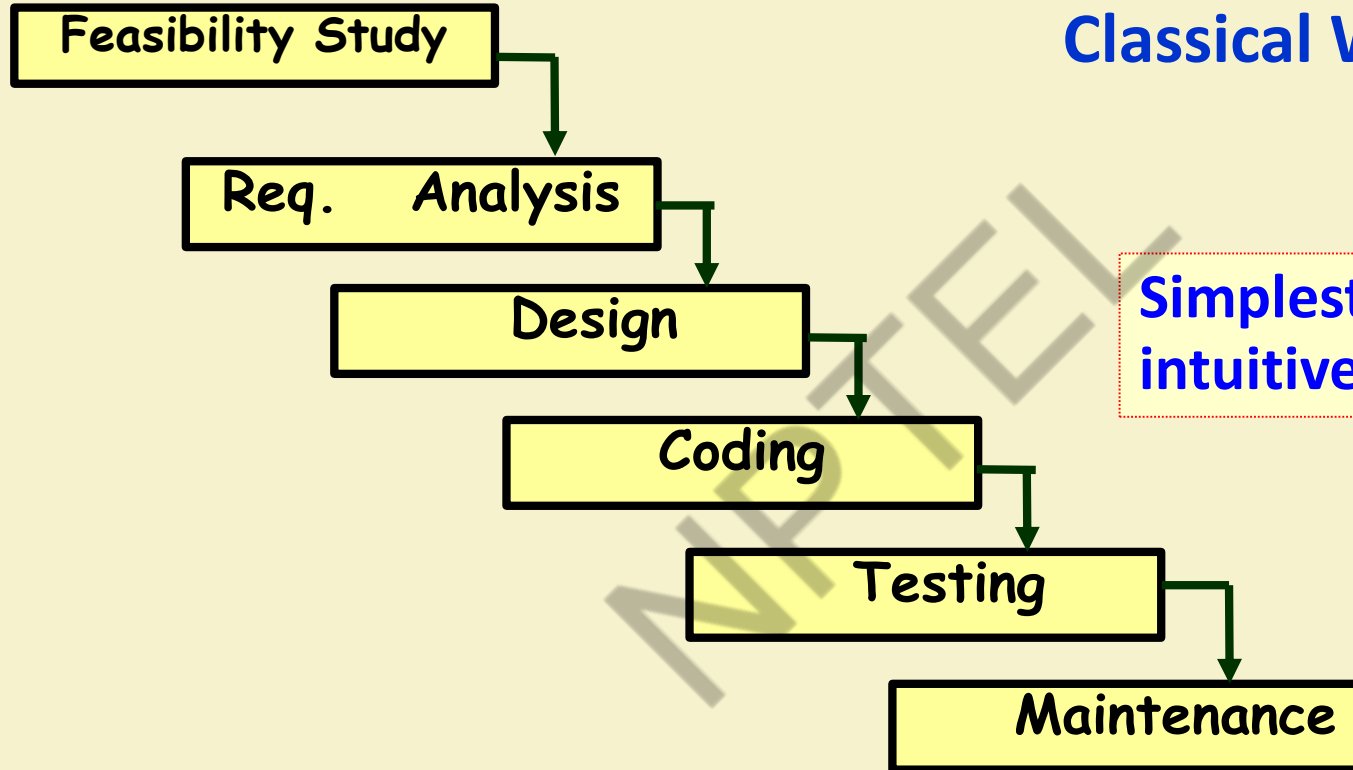
Software Life Cycle

Classical Waterfall Model

- Classical waterfall model divides life cycle into following phases:
 - Feasibility study,
 - Requirements analysis and specification,
 - Design,
 - Coding and unit testing,
 - Integration and system testing,
 - Maintenance.



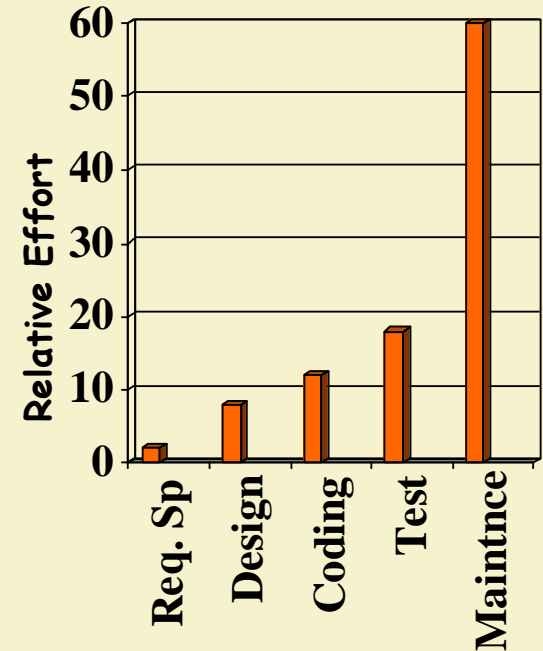
Classical Waterfall Model



Simplest and most intuitive

Relative Effort for Phases

- Phases between feasibility study and testing
 - Called **development phases**.
- Among all life cycle phases
 - **Maintenance phase consumes maximum effort.**
- Among development phases,
 - Testing phase consumes the maximum effort.



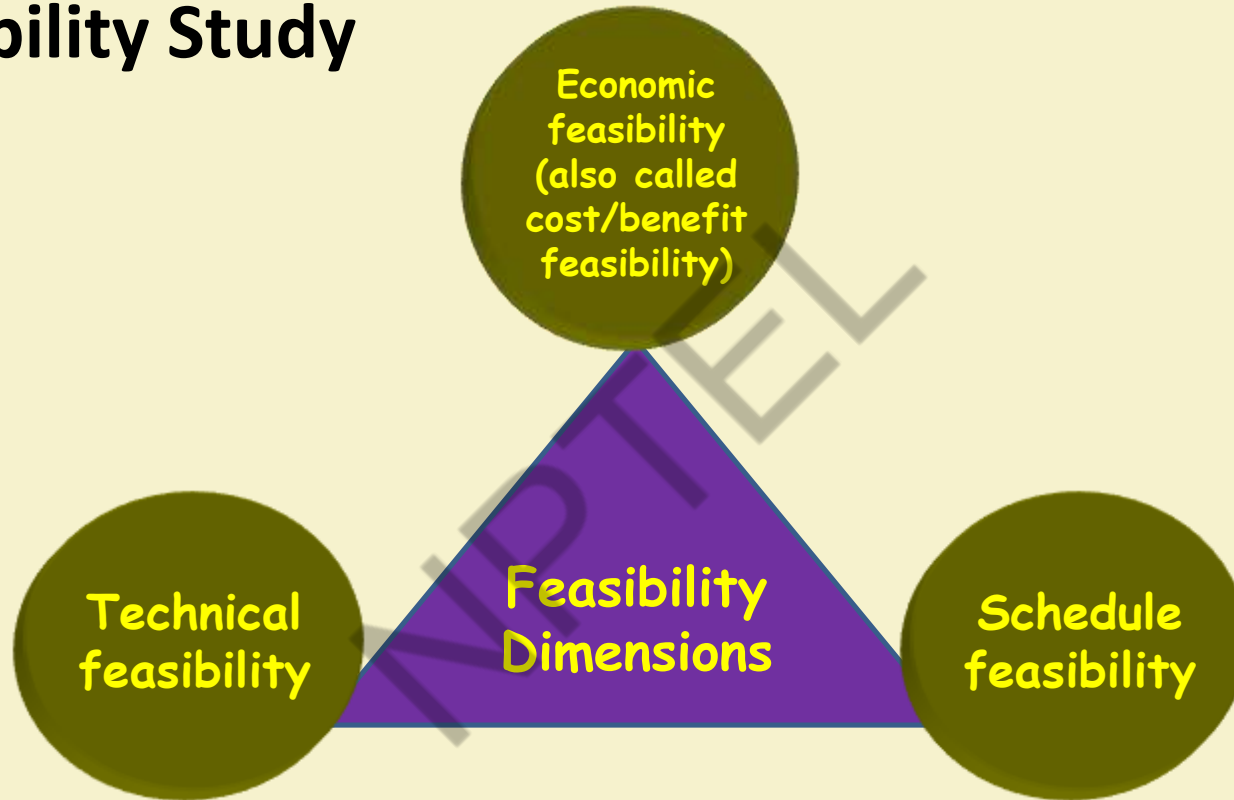
- Most organizations usually define:
 - Standards on the outputs (deliverables) produced at the end of every phase
 - Entry and exit criteria for every phase.
- They also prescribe methodologies for:
 - Specification,
 - Design,
 - Testing,
 - Project management, etc.

Process Model

Classical Waterfall Model (CONT.)

- The guidelines and methodologies of an organization:
 - Called the organization's **software development methodology**.
- Software development organizations:
 - Expect fresh engineers to master the organization's software development methodology.

Feasibility Study



- Main aim of feasibility study: **determine whether developing the software is:**
 - **Financially worthwhile**
 - **Technically feasible.**
- Roughly understand what customer wants:
 - **Data which would be input to the system,**
 - **Processing needed on these data,**
 - **Output data to be produced by the system,**
 - **Various constraints on the behavior of the system.**

Feasibility Study

First Step

- **SPF Scheme for CFL**
- CFL has a large number of employees, exceeding 50,000.
- Majority of these are casual labourers
- Mining being a risky profession:
 - Casualties are high
- Though there is a PF:
 - But settlement time is high
- There is a need of SPF:
 - For faster disbursement of benefits

Case Study

Feasibility: Case Study

- Manager visits main office, finds out the main functionalities required
- Visits mine site, finds out the data to be input
- Suggests alternate solutions
- Determines the best solution
- Presents to the CFL Officials
- Go/No-Go Decision

Activities During Feasibility Study

- Work out an overall understanding of the problem.
- Formulate different solution strategies.
- Examine alternate solution strategies in terms of:
 - resources required,
 - cost of development, and
 - development time.

- Perform a cost/benefit analysis:
 - Determine which solution is the best.
 - May also find that none of the solutions is feasible due to:

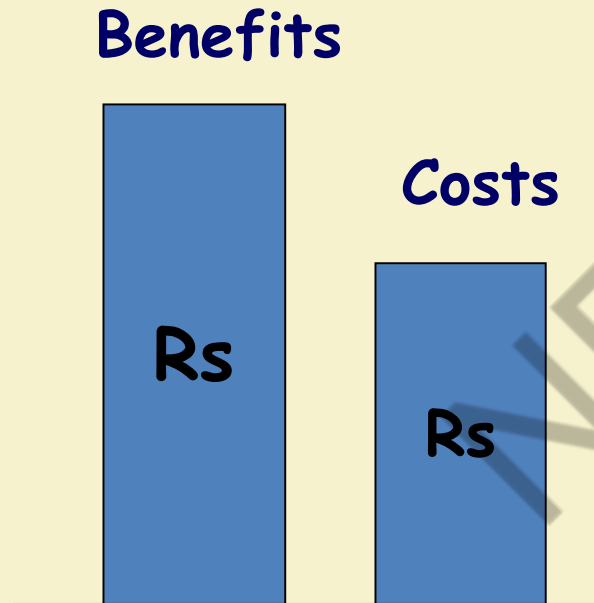
Activities during Feasibility Study

- high cost,
- resource constraints,
- technical reasons.

Cost benefit analysis (CBA)

- Need to identify all costs --- these could be:
 - Development costs
 - Set-up
 - Operational costs
- Identify the value of benefits
- Check benefits are greater than costs

The business case



- Benefits of delivered project must outweigh costs
- Costs include:
 - Development
 - Operation
- Benefits:
 - Quantifiable
 - Non-quantifiable

The business case

- Feasibility studies should help write a 'business case'
- Should provide a justification for starting the project
- Should show that the benefits of the project will exceed:
 - **Various costs**
- Needs to take account of business risks

Writing an Effective Business Case

1. Executive summary

2. Project background:

- The focus must be on what, exactly, the project is undertaking, and should not be confused with what might be a bigger picture.

3. Business opportunity

- What difference will it make?
- What if we don't do it?

4. Costs

- Should include the cost of development, implementation, training, change management, and operations.

5. Benefits

- Benefits usually presented in terms of revenue generation and cost reductions.

6. Risks

- Identify risks.
- Explain how these will be managed.

Feasibility Study

Req. Analysis

Design

Coding

Testing

Maintenance

Classical Waterfall Model

Requirements Analysis and Specification

- Aim of this phase:
 - Understand the exact requirements of the customer,
 - Document them properly.
- Consists of two distinct activities:
 - Requirements gathering and analysis
 - Requirements specification.

Requirements Analysis and Specification

- Gather requirements data from the customer:
 - Analyze the collected data to understand what customer wants
- Remove requirements problems:
 - Inconsistencies
 - Anomalies
 - Incompleteness
- Organize into a Software Requirements Specification (SRS) document.

Requirements Gathering

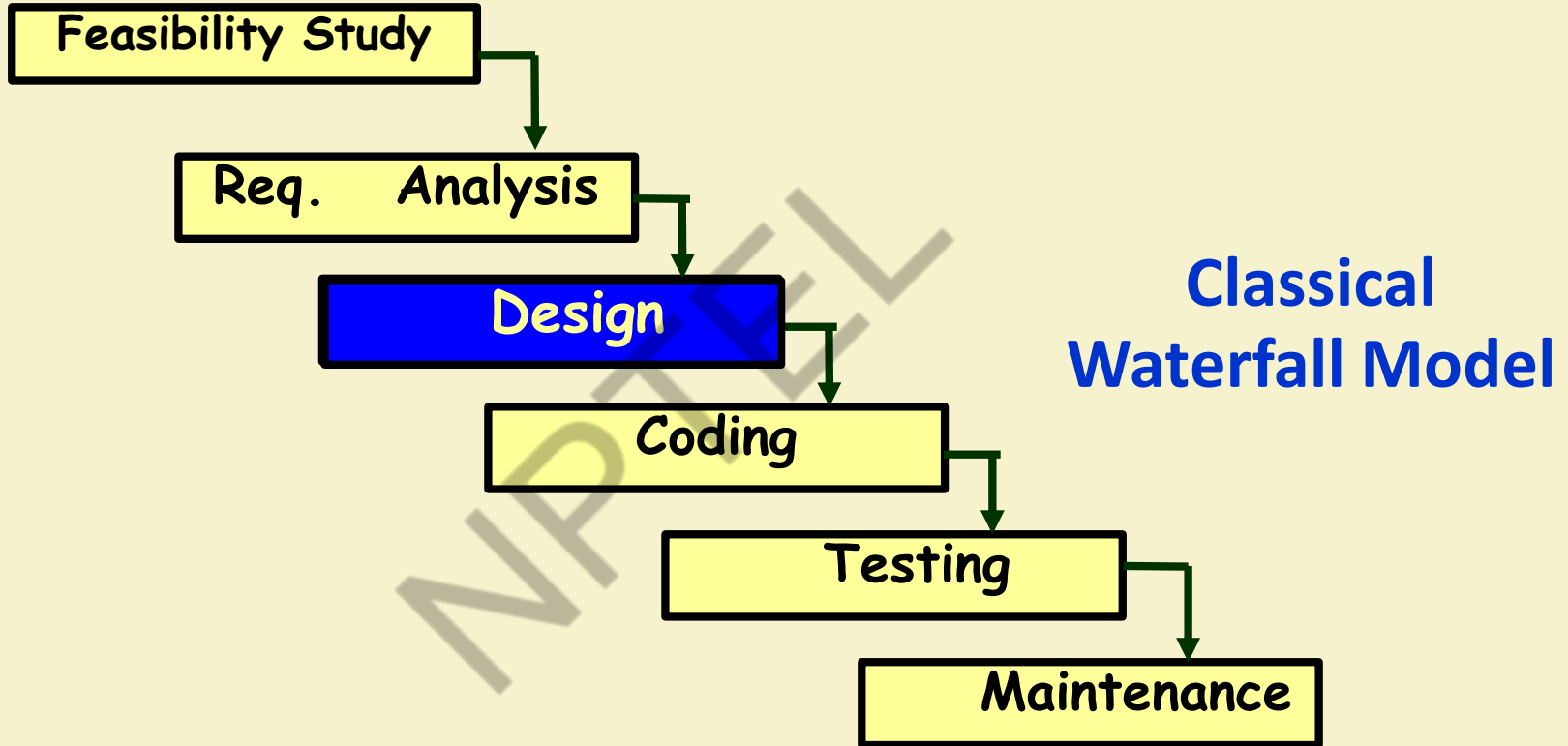
- Gathering relevant data:
 - Usually collected from the end-users through interviews and discussions.
 - **Example:** for a business accounting software:
 - Interview all the accountants of the organization to find out their requirements.

Requirements Analysis (Cont...)

- The data you initially collect from the users:
 - Usually contain several contradictions and ambiguities.
 - Why?
 - **Each user typically has only a partial and incomplete view of the system.**

Requirements Analysis (Cont...)

- Ambiguities and contradictions:
 - must be identified
 - resolved by discussions with the customers.
- Next, requirements are organized:
 - into a **Software Requirements Specification (SRS)** document.



Design

- During design phase requirements specification is transformed into :
 - A form suitable for implementation in some programming language.
- Two commonly used design approaches:
 - Traditional approach,
 - Object oriented approach

Traditional Design Approach

- Consists of two activities:
 - Structured analysis (typically carried out by using DFD)
 - Structured design

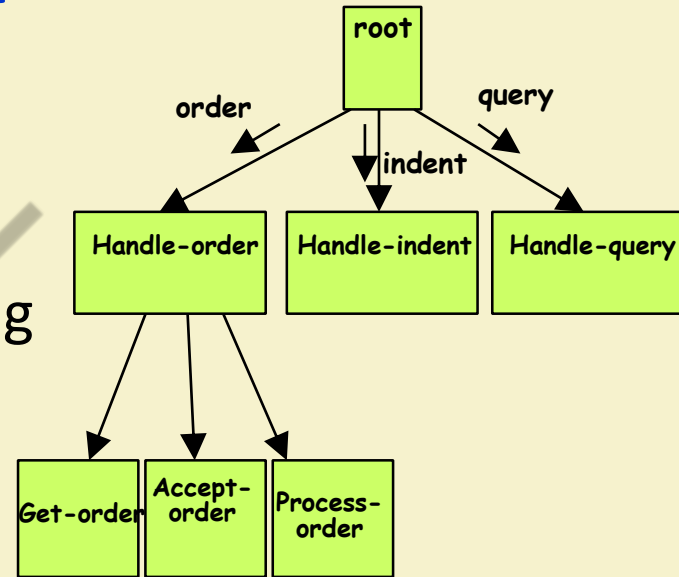
Structured Design

- High-level design:

- decompose the system into **modules**,
- represent invocation relationships among the modules.

- Detailed design:

- different modules designed in greater detail:
 - data structures and algorithms for each module are designed.

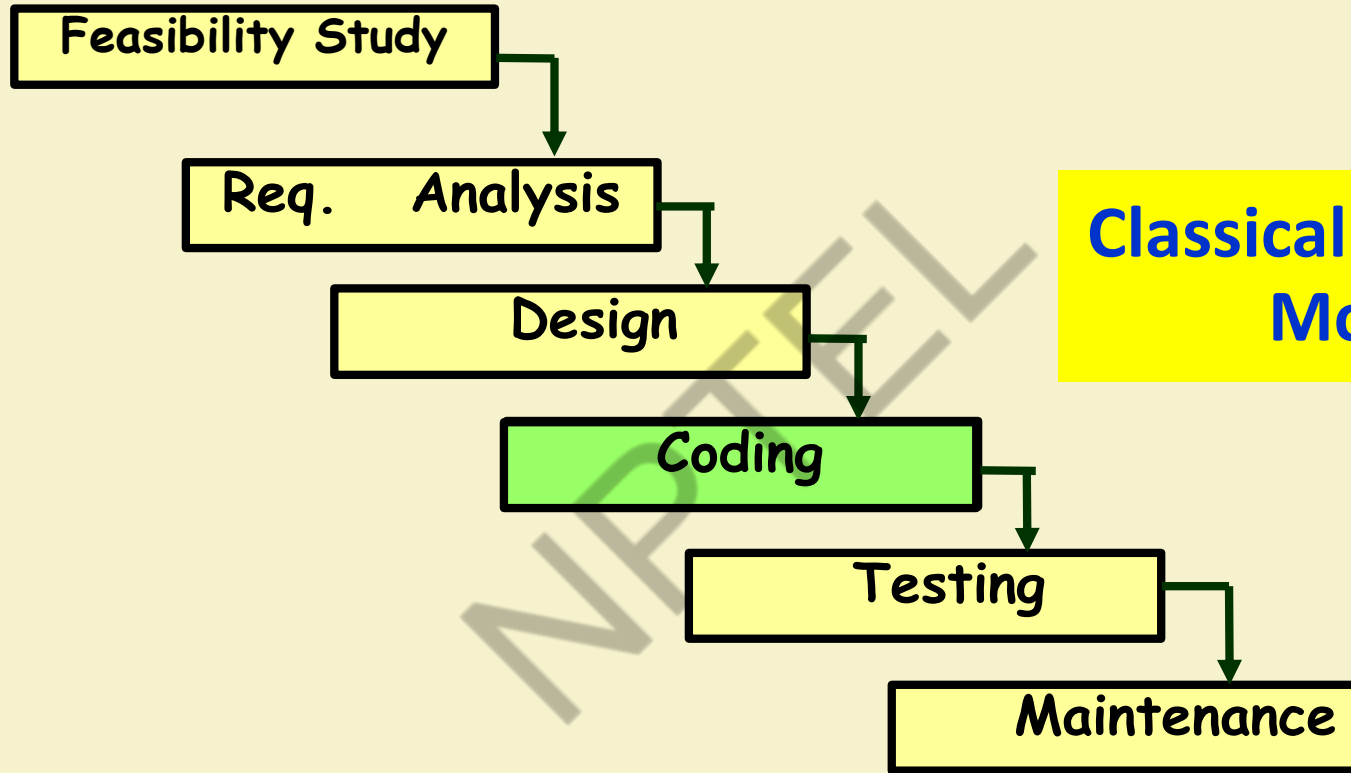


- First identify various objects (real world entities) occurring in the problem:
 - Identify the relationships among the objects.
 - For example, the objects in a pay-roll software may be:
 - employees,
 - managers,
 - pay-roll register,
 - Departments, etc.

Object-Oriented Design

Object Oriented Design (CONT.)

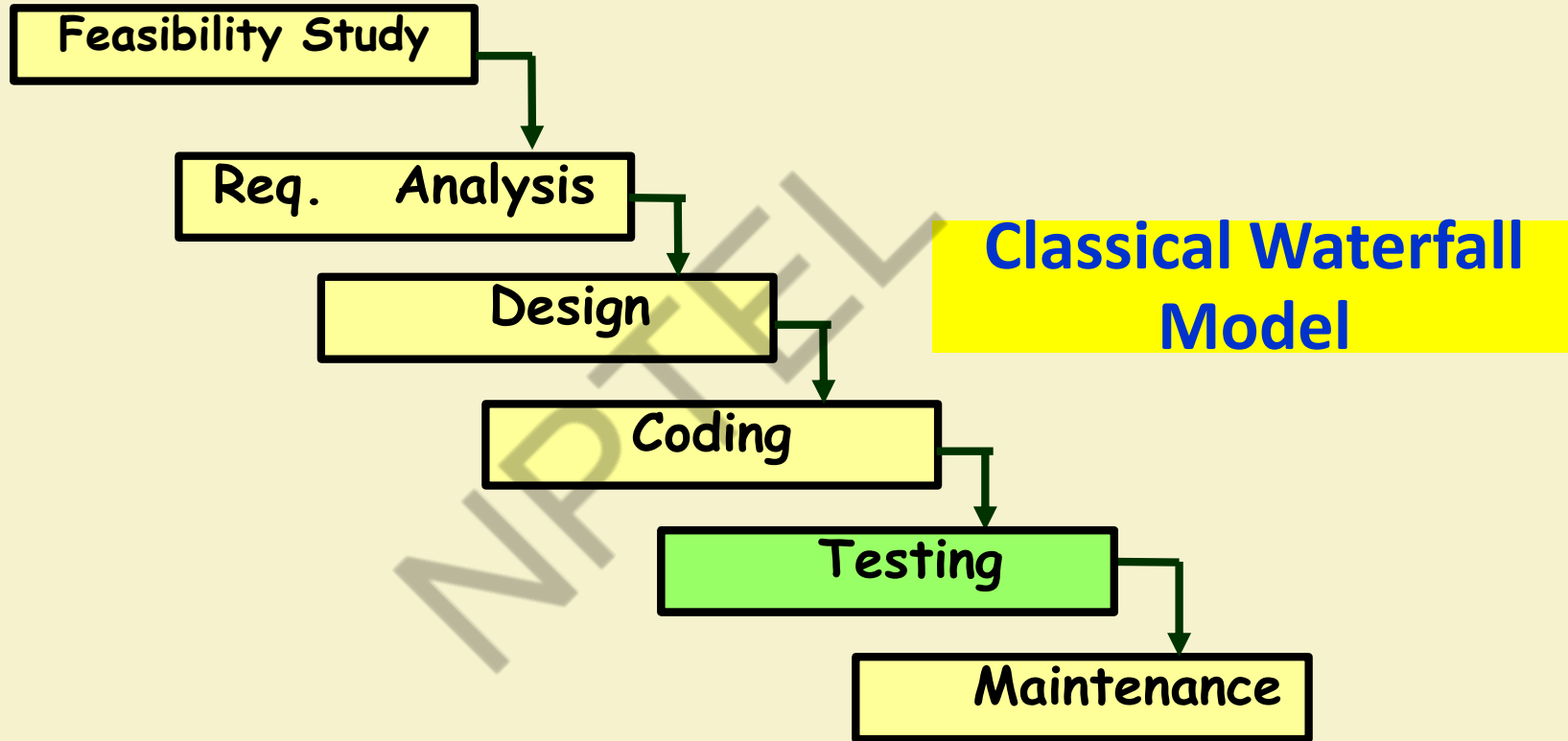
- Object structure:
 - Refined to obtain the detailed design.
- OOD has several advantages:
 - Lower development effort,
 - Lower development time,
 - Better maintainability.



Classical Waterfall Model

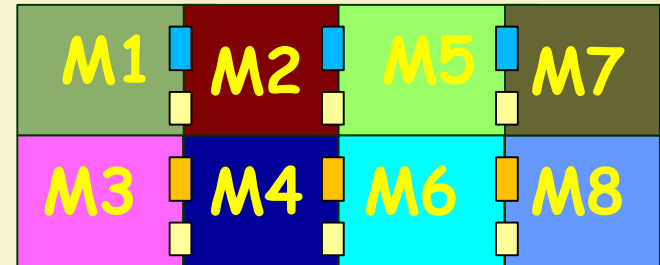
Coding and Unit Testing

- During this phase:
 - Each module of the design is coded,
 - Each module is unit tested
 - That is, tested independently as a stand alone unit, and debugged.
 - Each module is documented.



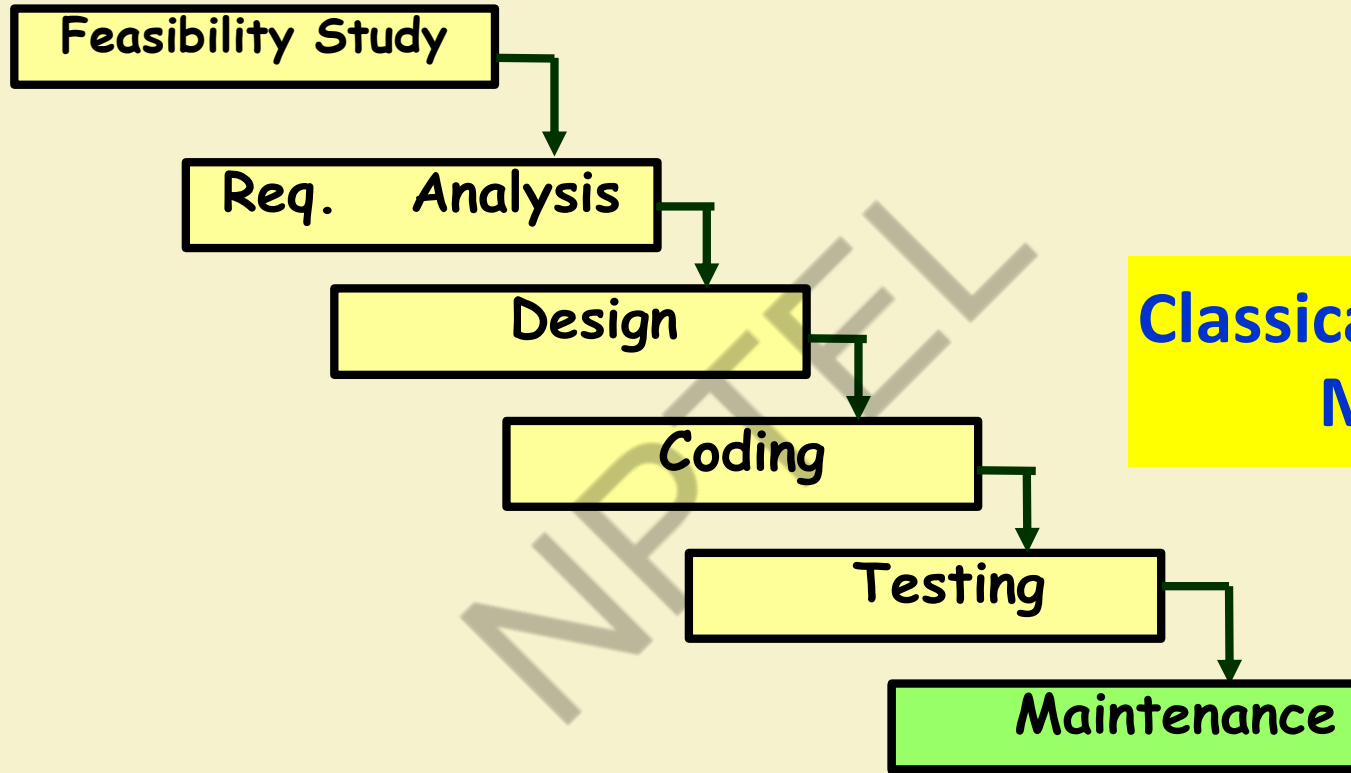
Integration and System Testing

- Different modules are integrated in a planned manner:
 - Modules are usually integrated through a number of steps.
- During each integration step,
 - the partially integrated system is tested.



System Testing

- After all the modules have been successfully integrated and tested:
 - System testing is carried out.
- Goal of system testing:
 - Ensure that the developed system functions according to its requirements as specified in the SRS document.



Classical Waterfall Model

Maintenance

- Maintenance of any software:
 - Requires much more effort than the effort to develop the product itself.
 - Development effort to maintenance effort is typically 40:60.

Types of Maintenance?

- **Corrective maintenance:**
 - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance:**
 - Improve implementation of the system
 - enhance functionalities of the system.
- **Adaptive maintenance:**
 - Port software to a new environment,
 - e.g. to a new computer or to a new operating system.

Iterative Waterfall Model

- Classical waterfall model is idealistic:
 - Assumes that no defect is introduced during any development activity.
 - In practice:
 - Defects do get introduced in almost every phase of the life cycle.

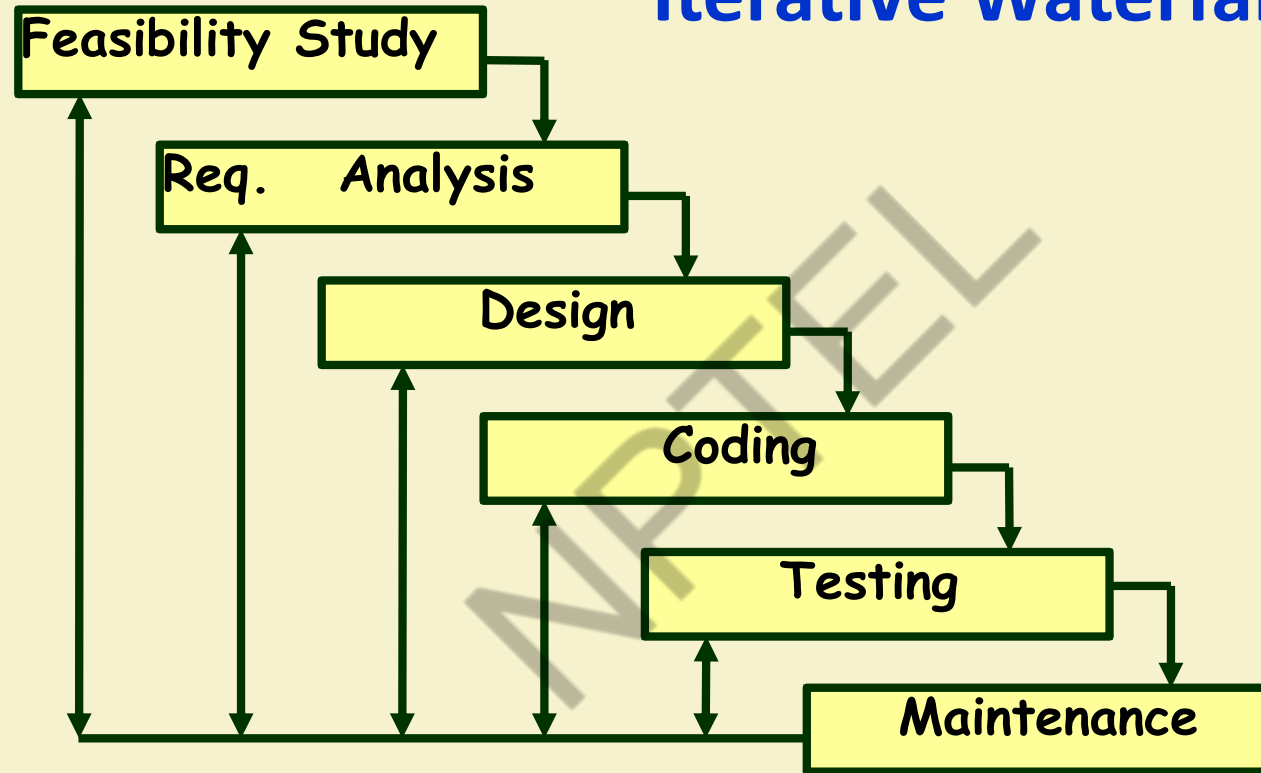
Iterative Waterfall Model (CONT.)

- Defects usually get detected much later in the life cycle:
 - For example, a design defect might go unnoticed till the coding or testing phase.
 - The later the phase in which the defect gets detected, the more expensive is its removal --- why?**

Iterative Waterfall Model (CONT.)

- Once a defect is detected:
 - The phase in which it occurred needs to be reworked.
 - Redo some of the work done during that and all subsequent phases.
- Therefore need feedback paths in the classical waterfall model.

Iterative Waterfall Model (CONT.)



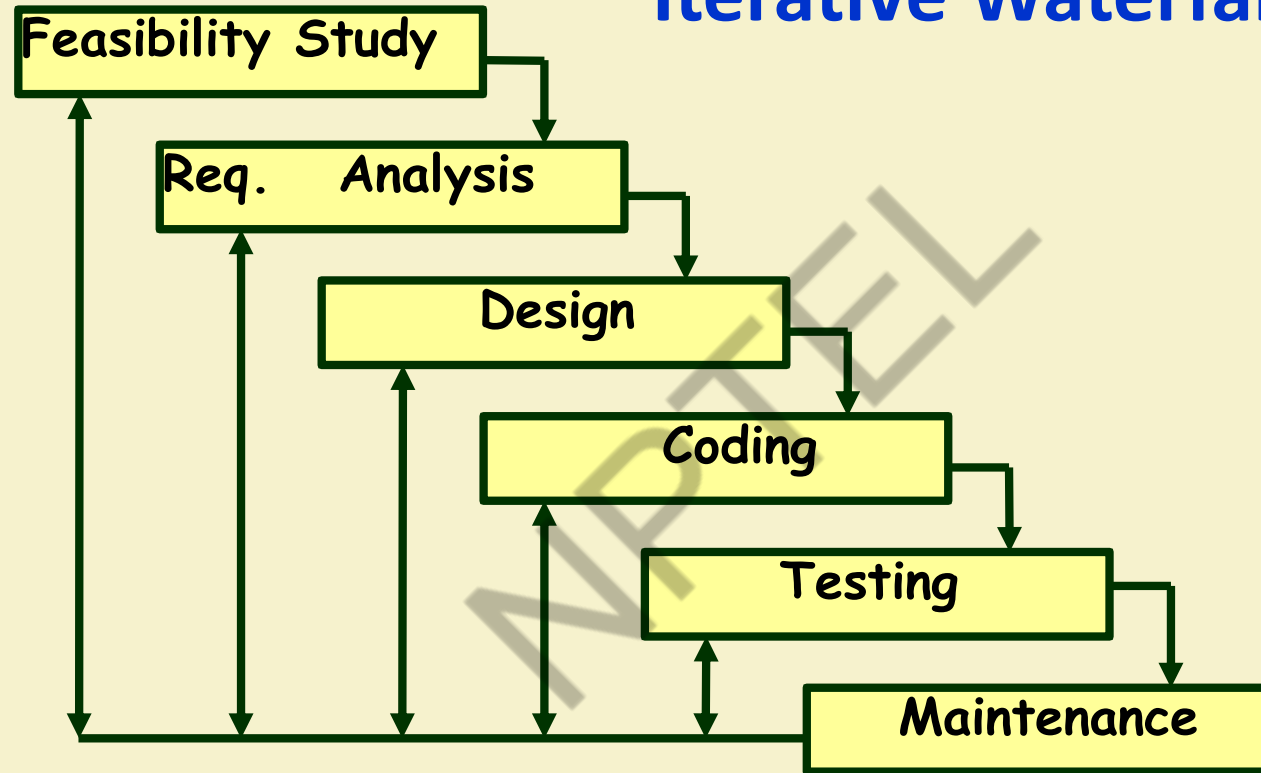
Phase Containment of Errors (Cont...)

- **Errors should be detected:**
 - **In the same phase in which they are introduced.**
- **For example:**
 - If a design problem is detected in the design phase itself,
 - The problem can be taken care of much more easily
 - Than say if it is identified at the end of the integration and system testing phase.

Phase Containment of Errors

- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible:
 - is known as **phase containment of errors.**
- Iterative waterfall model is by far the most widely used model.
 - Almost every other model is derived from the waterfall model.

Iterative Waterfall Model (CONT.)



Waterfall Strengths

- Easy to understand, easy to use, especially by inexperienced staff
- Milestones are well understood by the team
- Provides requirements stability during development
- Facilitates strong management control (plan, staff, track)

Waterfall Deficiencies

- All requirements must be known upfront – **in most projects requirement change occurs after project start**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Development team have experience with similar projects

Classical Waterfall Model (CONT.)

- Irrespective of the life cycle model actually followed:
 - The documents should reflect a classical waterfall model of development.
 - Facilitates comprehension of the documents.**

Classical Waterfall Model (CONT.)

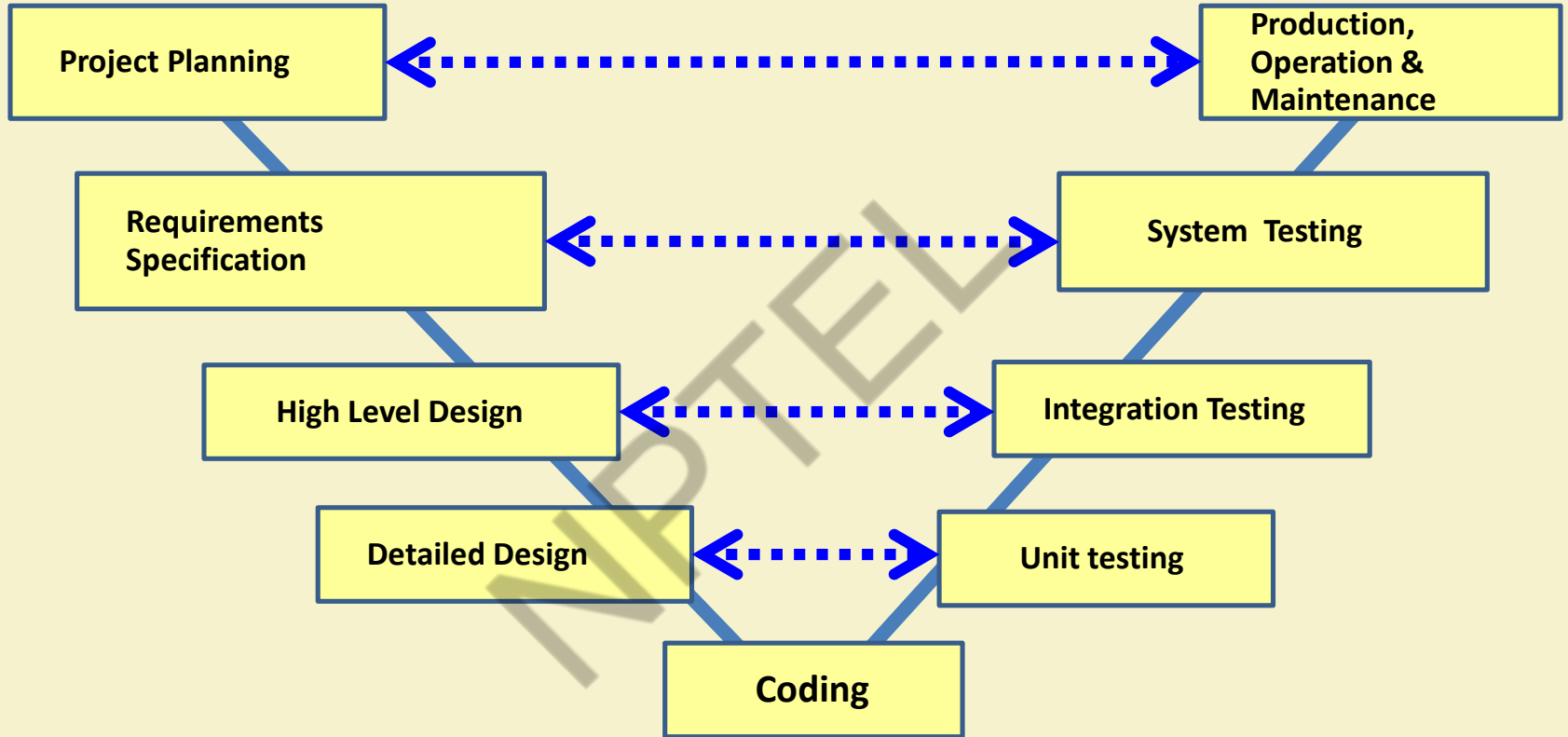
- Metaphor of mathematical theorem proving:
 - A mathematician presents a proof as a single chain of deductions,
 - Even though the proof might have come from a convoluted set of partial attempts, blind alleys and backtracks.

V Model



V Model

- It is a variant of the Waterfall
 - emphasizes verification and validation
 - V&V activities are spread over the entire life cycle.
- In every phase of development:
 - Testing activities are planned in parallel with development.



V Model Steps

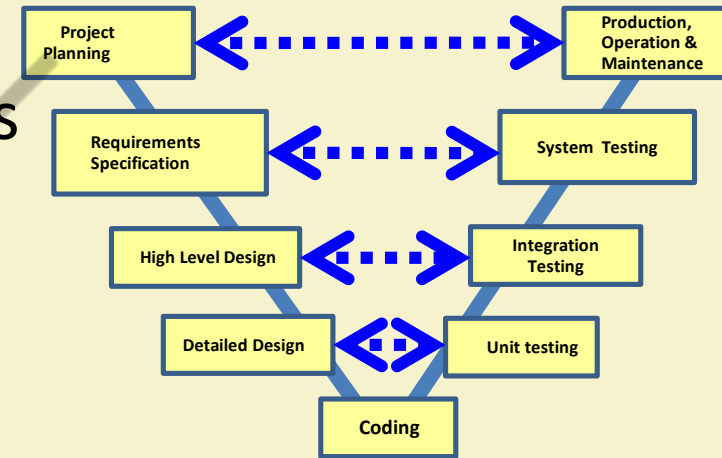
- Planning
- Requirements Analysis and Specification
 - System test design
- High-level Design
 - Integration Test design
- Detailed Design
 - Unit test design

V Model: Strengths

- Starting from early stages of software development:
 - **Emphasizes planning for verification and validation of the software**
- Each deliverable is made testable
- Easy to use

V Model Weaknesses

- Does not support overlapping of phases
- Does not handle iterations or phases
- Does not easily accommodate later changes to requirements
- Does not provide support for effective risk handling



When to use V Model

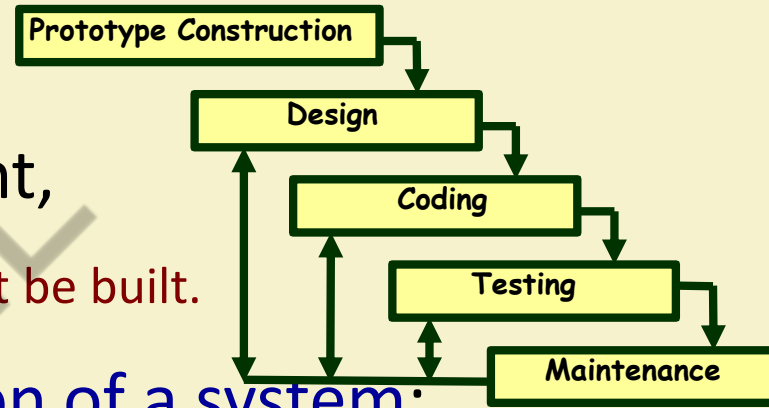
- Natural choice for systems requiring high reliability:
 - Embedded control applications, safety-critical software
- All requirements are known up-front
- Solution and technology are known

Prototyping Model



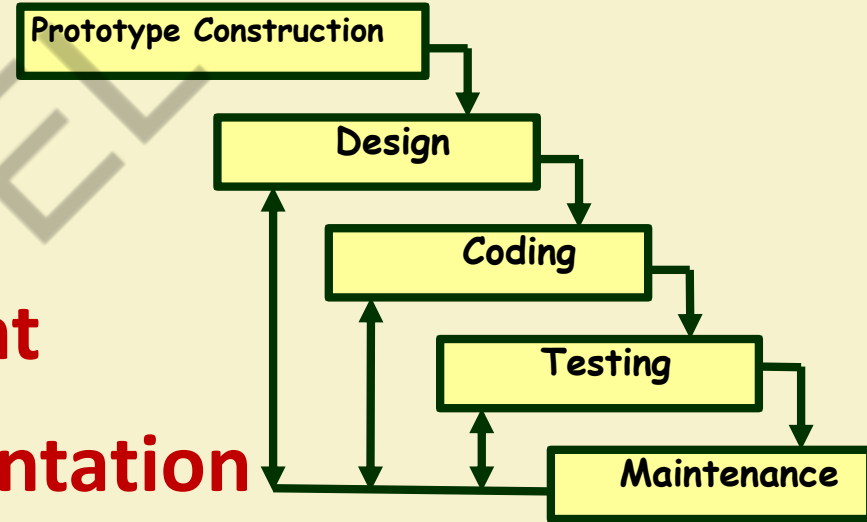
Prototyping Model

- A derivative of waterfall model.
- Before starting actual development,
 - A working prototype of the system should first be built.
- A prototype is a toy implementation of a system:
 - Limited functional capabilities,
 - Low reliability,
 - Inefficient performance.



Reasons for prototyping

- **Learning by doing:** useful where requirements are only partially known
- **Improved communication**
- **Improved user involvement**
- **Reduced need for documentation**
- **Reduced maintenance costs**



Reasons for Developing a Prototype

- **Illustrate to the customer:**

- input data formats, messages, reports, or interactive dialogs.

- **Examine technical issues associated with product development:**

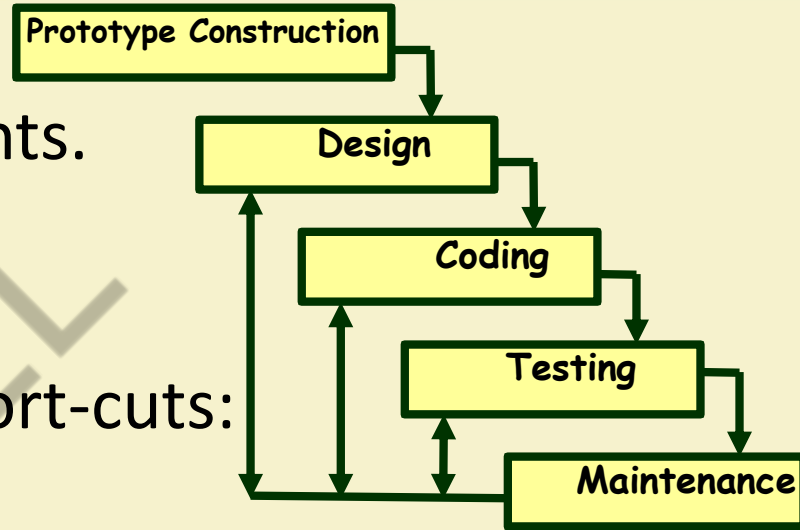
- Often major design decisions depend on issues like:

- Response time of a hardware controller,
 - Efficiency of a sorting algorithm, etc.

Prototyping Model (CONT.)

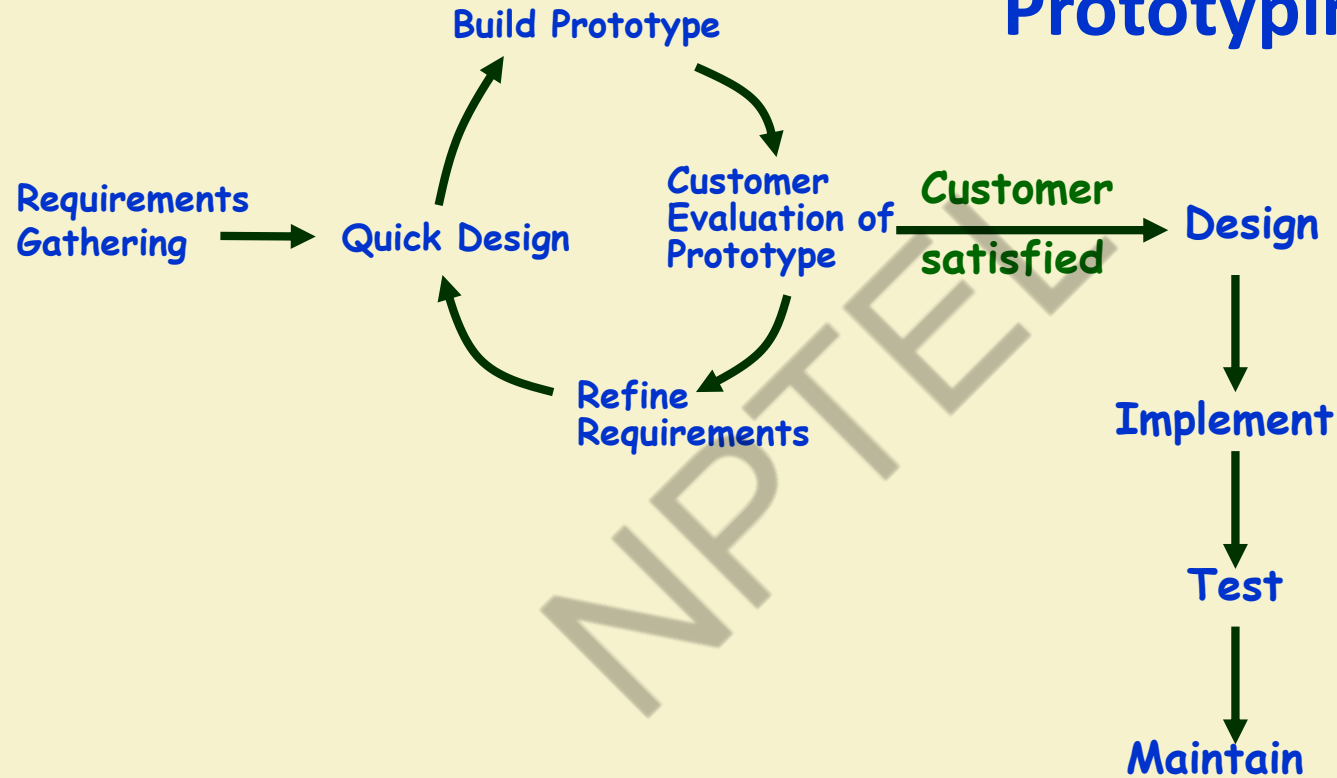
- Another reason for developing a prototype:
 - **It is impossible to “get it right” the first time,**
 - We must plan to throw away the first version:
 - If we want to develop a good software.

Prototyping Model



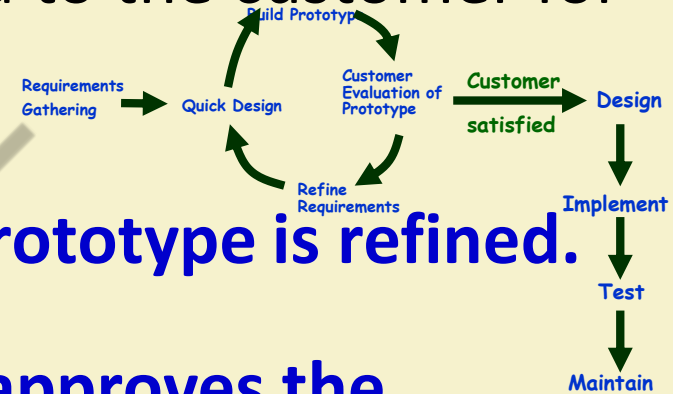
- Start with approximate requirements.
- Carry out a quick design.
- Prototype is built using several short-cuts:
 - Short-cuts might involve:
 - Using inefficient, inaccurate, or dummy functions.
 - A table look-up rather than performing the actual computations.

Prototyping Model (CONT.)



Prototyping Model (CONT.)

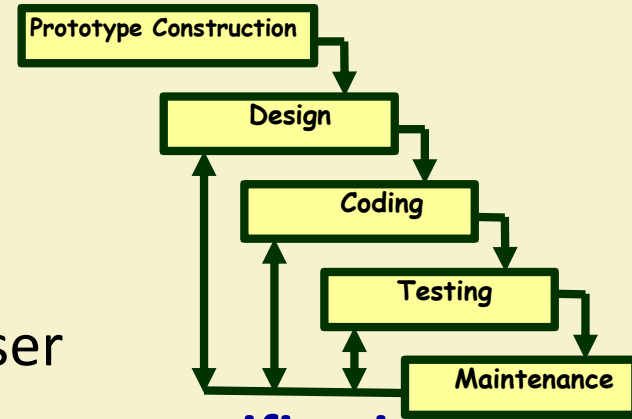
- The developed prototype is submitted to the customer for his evaluation:



- Based on the user feedback, the prototype is refined.**
 - This cycle continues until the user approves the prototype.**
- The actual system is developed using the waterfall model.

Prototyping Model

- Requirements analysis and specification phase becomes redundant:
 - Final working prototype (incorporating all user feedbacks) serves as an **animated requirements specification**.
- **Design and code for the prototype is usually thrown away:**
 - However, experience gathered from developing the prototype helps a great deal while developing the actual software.



Prototyping Model (CONT.)

- Even though construction of a working prototype model involves additional cost --- **overall development cost usually lower for:**
 - Systems with unclear user requirements,
 - Systems with unresolved technical issues.
- Many user requirements get properly defined and technical issues get resolved:
 - These would have appeared later as change requests and resulted in incurring massive redesign costs.

Prototyping: advantages

- The resulting software is usually more usable
- User needs are better accommodated
- The design is of higher quality
- The resulting software is easier to maintain
- Overall, the development incurs less cost

Prototyping: disadvantages

- For some projects, it is expensive
- Susceptible to over-engineering:
 - Designers start to incorporate sophistications that they could not incorporate in the prototype.

Major difficulties of Waterfall-Based Models

1. Difficulty in accommodating change requests during development.
 - **40% of the requirements change during development**
2. High cost incurred in developing custom applications.
3. **“Heavy weight processes.”**

Major difficulties of Waterfall-Based Life Cycle Models

- Requirements for the system are determined at the start:
 - Are assumed to be fixed from that point on.
 - Long term planning is made based on this.

“... the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. ...this assumption is fundamentally wrong and many software acquisition problems spring from this...”

Frederick Brooks

Incremental Model

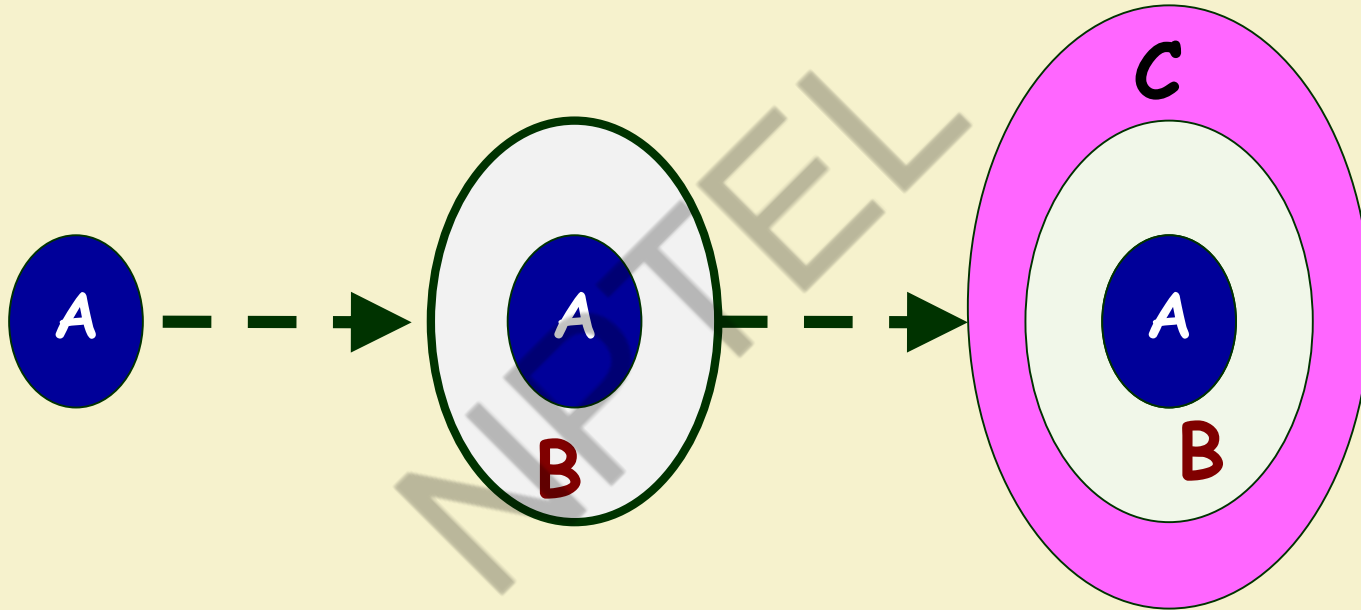


- “The basic idea... take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system... Start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions. At each version design modifications are made along with adding new functional capabilities. “ **Victor Basili**

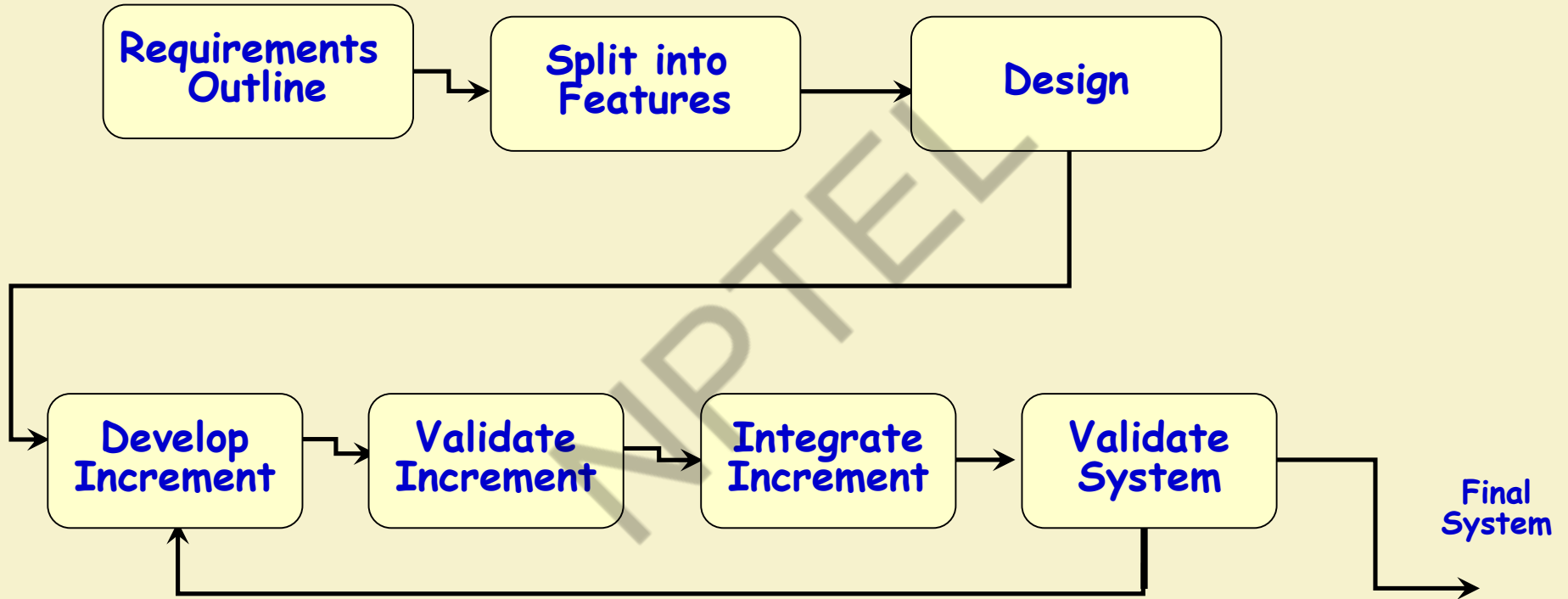
Incremental and Iterative Development (IID)

- **Key characteristics**
 - Builds system incrementally
 - Consists of a planned number of iterations
 - Each iteration produces a working program
- **Benefits**
 - Facilitates and manages changes
- **Foundation of agile techniques and the basis for**
 - Rational Unified Process (RUP)
 - Extreme Programming (XP)

Customer's Perspective



Incremental Model



Incremental Model: Requirements

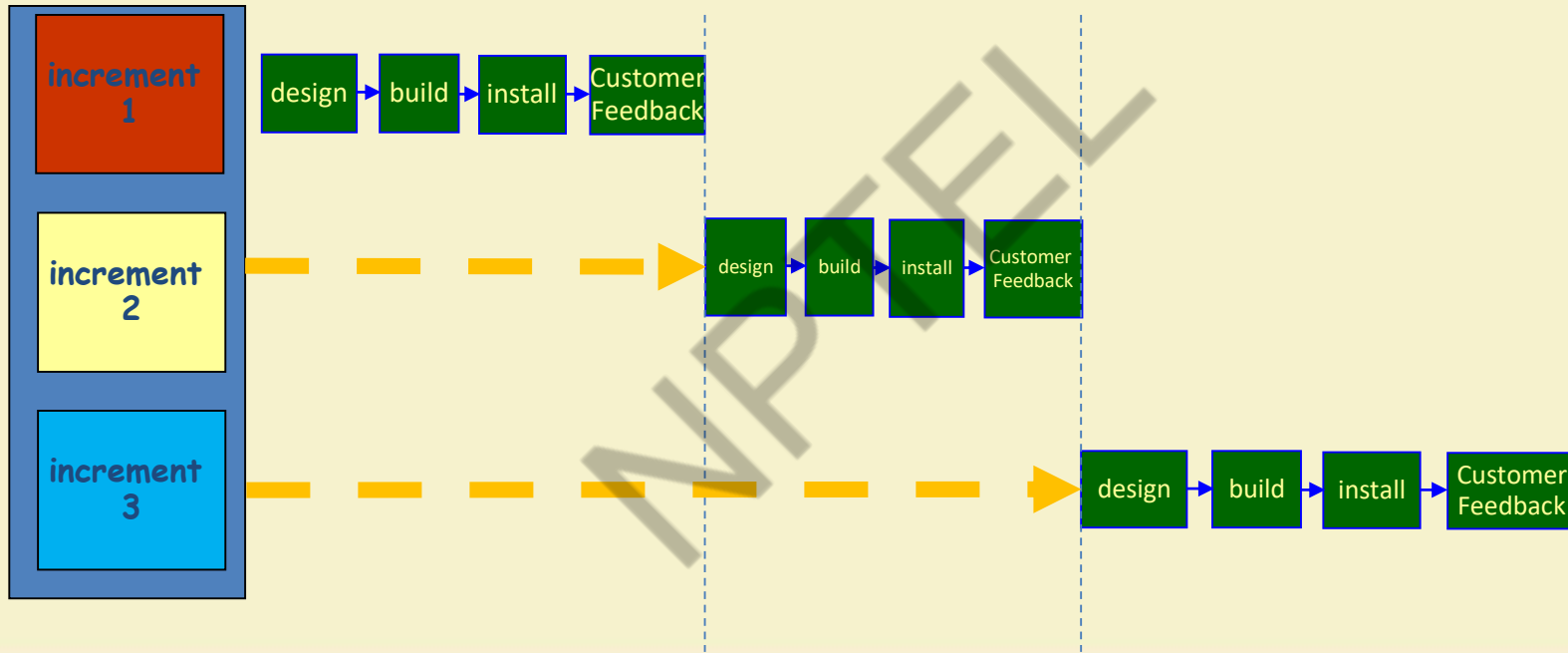
Split into
Features

Requirements: High Level Analysis											
Slice	Slice	Slice	Slice	Slice	Slice	Slice	Slice	Slice	Slice	Slice	Slice

Incremental Model

- Waterfall: single release
- Iterative: many releases (increments)
 - **First increment: core functionality**
 - **Successive increments: add/fix functionality**
 - **Final increment: the complete product**
- Each iteration: a short mini-project with a separate lifecycle
 - e.g., waterfall

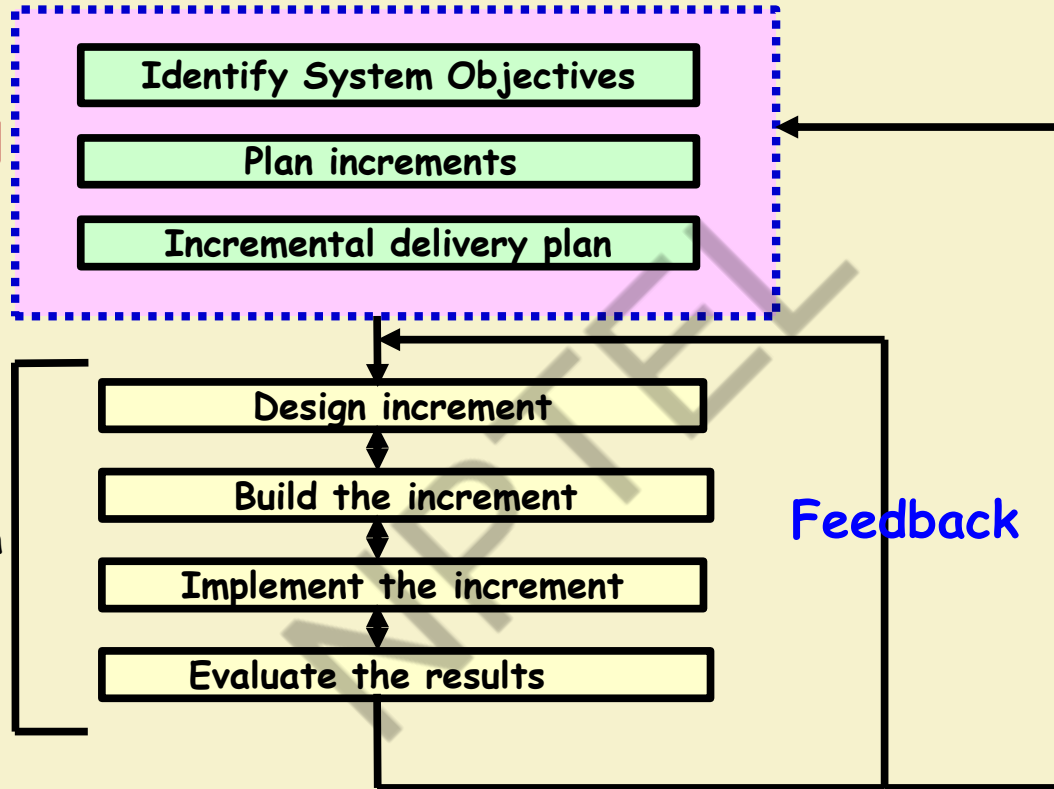
Incremental delivery



Planned
incremental
delivery

The
incremental
process

Repeat for each
increment



Which step first?

- Some steps will be pre-requisite because of physical dependencies
- Others may be in any order
- Value to cost ratios may be used
 - V/C where
 - V is a score 1-10 representing value to customer
 - C is a score 0-10 representing cost to developers

V/C ratios: an example

step	value	cost	ratio	
profit reports	9	2	4.5	2nd
online database	1	9	0.11	5th
ad hoc enquiry	5	5	1	4th
purchasing plans	9	4	2.25	3rd
profit-based pay for managers	9	1	9	1st

Evolutionary Model with Iterations



An Evolutionary and Iterative Development Process...

- Recognizes the reality of changing requirements
 - Capers Jones's research on 8000 projects: **40% of final requirements arrived after development had already begun**
- Promotes early risk mitigation:
 - Breaks down the system into mini-projects and focuses on the riskier issues first.
 - **“plan a little, design a little, and code a little”**
- Encourages all development participants to be involved earlier on, :
 - End users, Testers, integrators, and technical writers

Evolutionary Model with Iteration

- “A complex system will be most successful if implemented in small steps... “retreat” to a previous successful step on failure... opportunity to receive some feedback from the real world before throwing in all resources... and you can correct possible errors...” [Tom Glib in Software Metrics](#)

Evolutionary model with iteration

- Evolutionary iterative development implies that the requirements, plan, estimates, and solution evolve or are refined over the course of the iterations, rather than fully defined and “frozen” in a major up-front specification effort before the development iterations begin. Evolutionary methods are consistent with the pattern of unpredictable discovery and change in new product development.” **Craig Larman**

Evolutionary Model

- First develop the core modules of the software.
- The initial skeletal software is refined into increasing levels of capability: (Iterations)
 - By adding new functionalities in successive versions.

Activities in an Iteration

- Software developed over several “mini waterfalls”.
- The result of a single iteration:
 - Ends with delivery of some tangible code
 - An incremental improvement to the software --- leads to evolutionary development

Evolutionary Model with Iteration

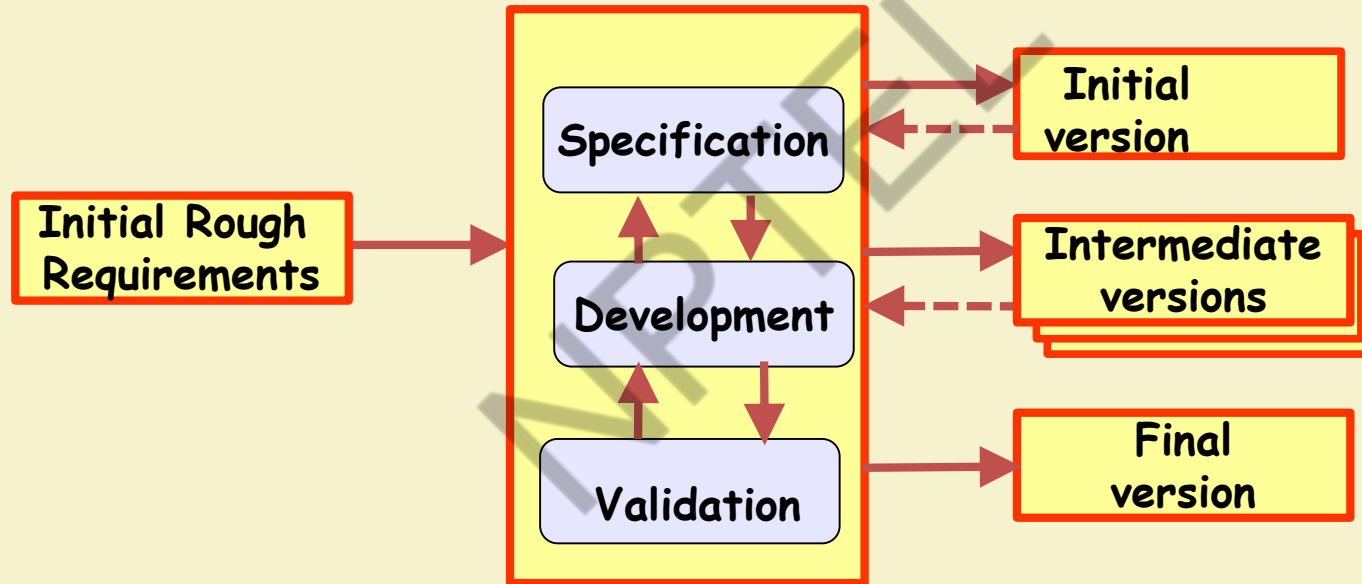
- Outcome of each iteration: tested, integrated, executable system
- Iteration length is short and fixed
 - Usually between 2 and 6 weeks
 - Development takes many iterations (for example: 10-15)
- Does not “freeze” requirements and then conservatively design :
 - Opportunity exists to modify requirements as well as the design...

Evolutionary Model (CONT.)

- Successive versions:
 - Functioning systems capable of performing some useful work.
 - A new release may include new functionality:
 - Also existing functionality in the current release might have been enhanced.

Evolutionary Model

- Evolves an initial implementation with user feedback:
 - Multiple versions until the final version.



Advantages of Evolutionary Model

- Users get a chance to experiment with a partially developed system:
 - Much before the full working version is released,
- **Helps finding exact user requirements:**
 - Software more likely to meet exact user requirements.
- **Core modules get tested thoroughly:**
 - Reduces chances of errors in final delivered software.

Advantages of evolutionary model

- Better management of complexity by developing one increment at a time.
- Better management of changing requirements.
- Can get customer feedback and incorporate them much more efficiently:
 - As compared when customer feedbacks come only after the development work is complete.

Advantages of Evolutionary Model with Iteration

- Training can start on an earlier release
 - customer feedback taken into account
- Frequent releases allow developers to fix unanticipated problems quicker.

Evolutionary Model: Problems

- **The process is intangible:**
 - No regular, well-defined deliverables.
- **The process is unpredictable:**
 - Hard to manage, e.g., scheduling, workforce allocation, etc.
- **Systems are rather poorly structured:**
 - Continual, unpredictable changes tend to degrade the software structure.
- **Systems may not even converge to a final version.**

RAD Model



Rapid Application Development (RAD) Model

- Sometimes referred to as the **rapid prototyping model**.
- Major aims:
 - Decrease the time taken and the cost incurred to develop software systems.
 - Facilitate accommodating change requests as early as possible:
 - Before large investments have been made in development and testing.

Important Underlying Principle

- A way to reduce development time and cost, and yet have flexibility to incorporate changes:
 - **Make only short term plans and make heavy reuse of existing code.**

Methodology

- Plans are made for one increment at a time.
 - The time planned for each iteration is called a **time box**.
- Each iteration (increment):
 - Enhances the implemented functionality of the application a little.

Methodology

- During each iteration,
 - A quick-and-dirty prototype-style software for some selected functionality is developed.
 - The customer evaluates the prototype and gives his feedback.
 - The prototype is refined based on the customer feedback.

How Does RAD Facilitate Faster Development?

- RAD achieves fast creation of working prototypes.
 - Through use of specialized tools.
- These specialized tools usually support the following features:
 - **Visual style of development.**
 - **Use of reusable components.**
 - **Use of standard APIs (Application Program Interfaces).**

For which Applications is RAD Suitable?

- Customized product developed for one or two customers only
- Performance and reliability are not critical.
- The system can be split into several independent modules.

For Which Applications RAD is Unsuitable?

- Few plug-in components are available
- High performance or reliability required
- No precedence for similar products exists
- The system cannot be modularized.

Prototyping versus RAD

- In prototyping model:
 - The developed prototype is primarily used to gain insights into the solution
 - Choose between alternatives
 - Elicit customer feedback.
- The developed prototype:
 - Usually thrown away.

Prototyping versus RAD

- In contrast:
 - In RAD the developed prototype evolves into deliverable software.
- RAD leads to faster development compared to traditional models:
 - However, the quality and reliability would possibly be poorer.

RAD versus Iterative Waterfall Model

- In the iterative waterfall model,
 - All product functionalities are developed together.
- In the RAD model on the other hand,
 - Product functionalities are developed incrementally through heavy code and design reuse.
 - Customer feedback is obtained on the developed prototype after each iteration:
 - Based on this the prototype is refined.

RAD versus Iterative Waterfall Model

- The iterative waterfall model:
 - Does not facilitate accommodating requirement change requests.
- Iterative waterfall model does have some important advantages:
 - Use of the iterative waterfall model leads to production of good documentation.
 - Also, the developed software usually has better quality and reliability than that developed using RAD.

RAD versus Evolutionary Model

- Incremental development:
 - Occurs in both evolutionary and RAD models.
- However, in RAD:
 - Each increment is a quick and dirty prototype,
 - Whereas in the evolutionary model each increment is systematically developed using the iterative waterfall model.
- Also, RAD develops software in shorter increments:
 - The incremental functionalities are fairly large in the evolutionary model.

Unified Process



Unified Process

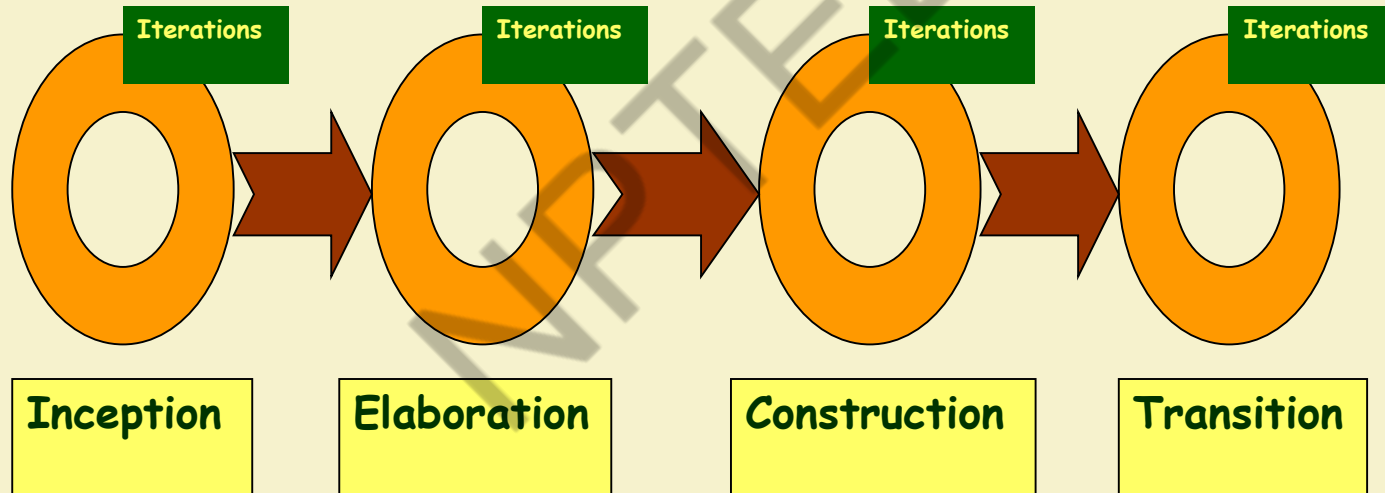
- Developed Ivar Jacobson, Grady Booch and James Rumbaugh
 - Incremental and iterative
- Rational Unified Process (RUP) is version tailored by Rational Software:
 - Acquired by IBM in February 2003.

Four Phases --- and iterative Development at Every phase

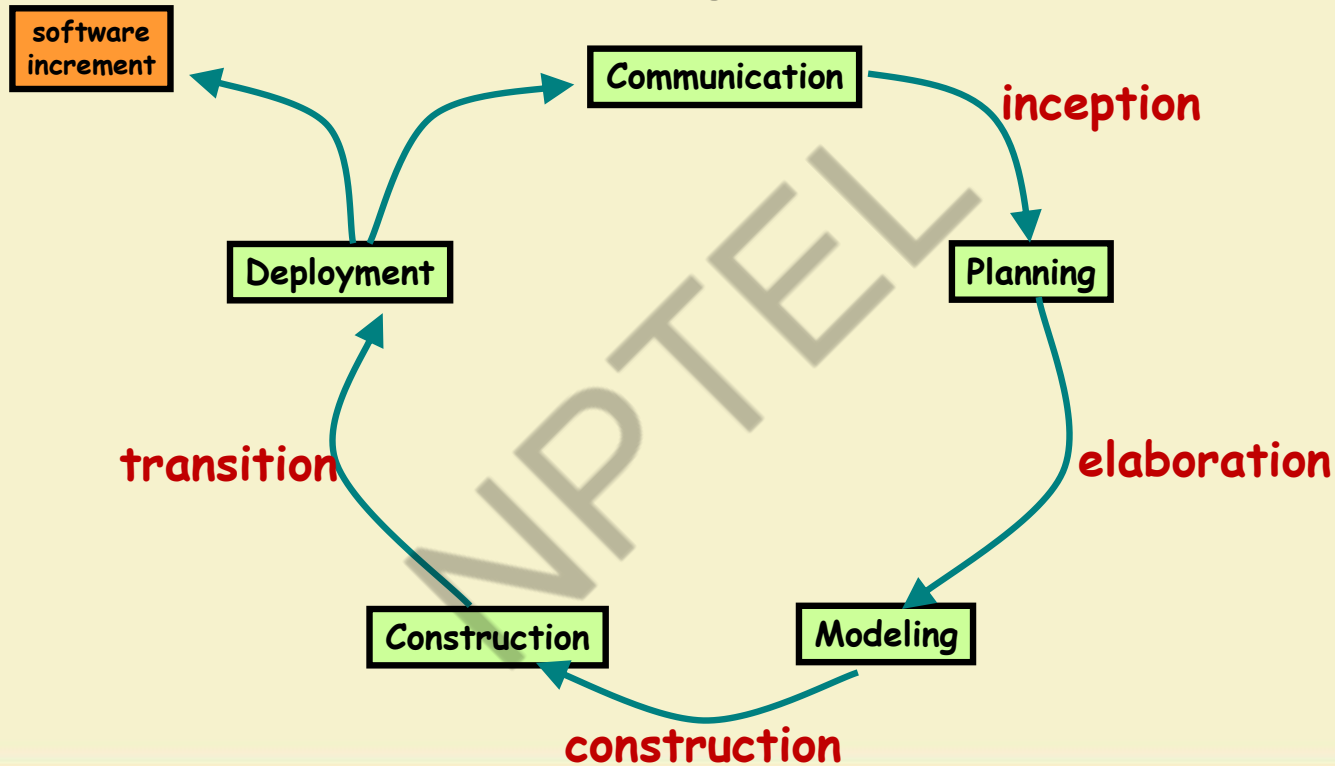
- Inception Phase
- Elaboration Phase
- Construction Phase
- Transition Phase

Unified Process Iterations in Phases

The duration of and iteration may vary from two weeks or less.



Unified process



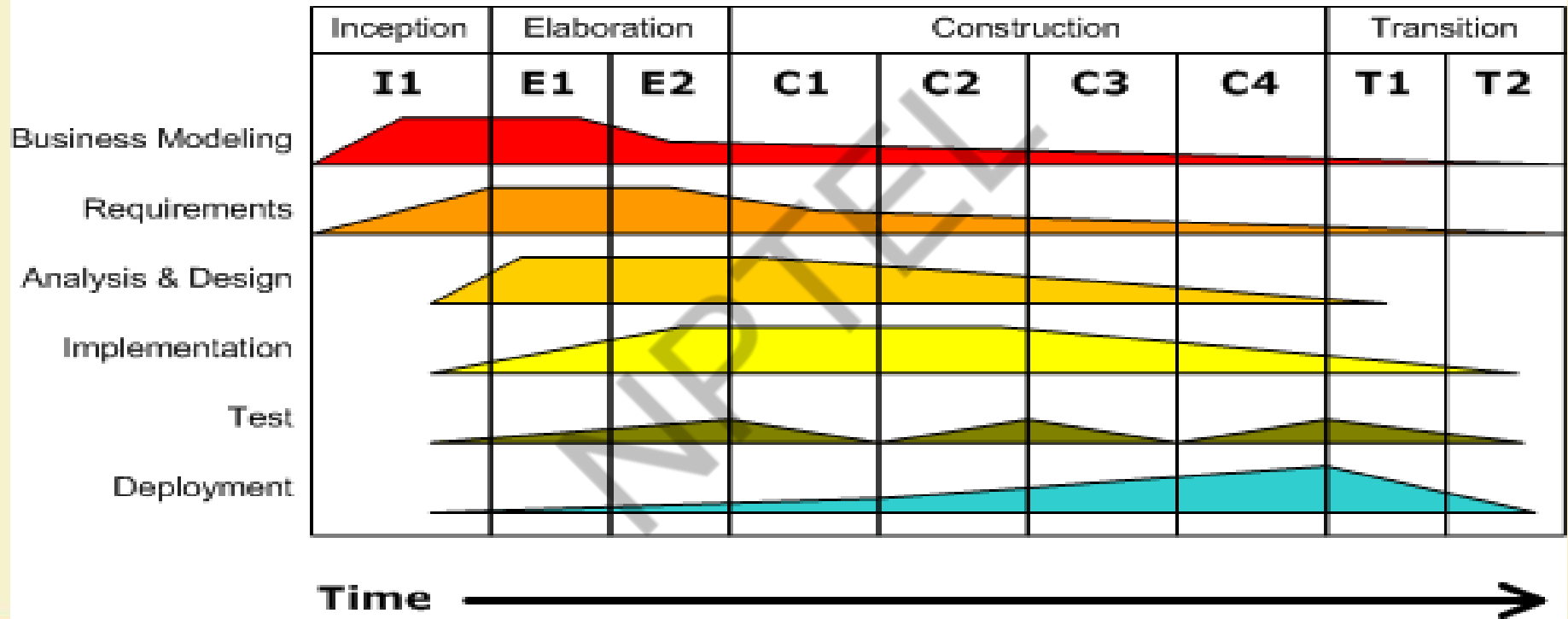
Unified process work products

<u>Inception phase</u> vision document initial use-case model initial business case initial risk list project plan prototype(s) ...	<u>Elaboration phase</u> use-case model requirements analysis model preliminary model revised risk list preliminary manual ...	<u>Construction phase</u> design model SW components test plan test procedure test cases user manual installation manual ...	<u>Transition phase</u> SW increment beta test reports user feedback ...
--	--	--	--

Structure of RUP Process

- Two dimensions.
- Horizontal axis:
 - Represents time and shows the lifecycle aspects of the process.
- Vertical axis:
 - Represents core process workflows.

Two dimensions of Unified Process



Inception activities

- Formulate scope of project
- Risk management, staffing, project plan
- Synthesize a candidate architecture.

- Initial requirements capture
- Cost Benefit Analysis
- Initial Risk Analysis
- Project scope definition
- Defining a candidate architecture
- Development of a disposable prototype
- Initial Use Case Model (10% - 20% complete)
- First pass Domain Model

Outcome of Inception Phase

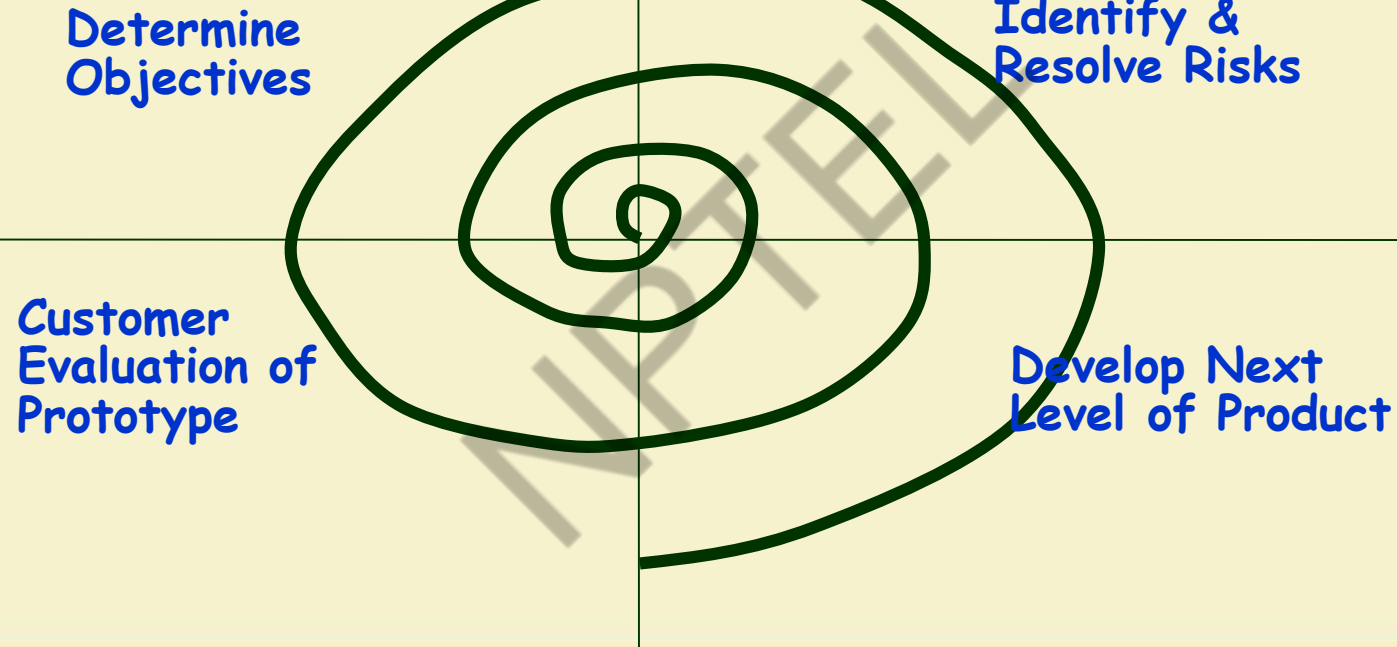
Spiral Model

- Proposed by Boehm in 1988.
- Each loop of the spiral represents a phase of the software process:
 - the innermost loop might be concerned with system feasibility,
 - the next loop with system requirements definition,
 - the next one with system design, and so on.
- There are no fixed phases in this model, the phases shown in the figure are just examples.

Spiral Model (CONT.)

- The team must decide:
 - how to structure the project into phases.
- Start work using some generic model:
 - add extra phases
 - for specific projects or when problems are identified during a project.
- Each loop in the spiral is split into four sectors (quadrants).

Spiral Model



Objective Setting (First Quadrant)

- Identify objectives of the phase,
- Examine the risks associated with these objectives.
 - Risk:
 - Any adverse circumstance that might hamper successful completion of a software project.
- Find alternate solutions possible.

Risk Assessment and Reduction (Second Quadrant)

- For each identified project risk,
 - a detailed analysis is carried out.
- Steps are taken to reduce the risk.
- For example, if there is a risk that requirements are inappropriate:
 - A prototype system may be developed.

Spiral Model (CONT.)

- Development and Validation (**Third quadrant**):
 - develop and validate the next level of the product.
- Review and Planning (**Fourth quadrant**):
 - review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral:
 - progressively more complete version of the software gets built.

- Subsumes all discussed models:
 - a single loop spiral represents waterfall model.
 - uses an evolutionary approach --
 - iterations over the spiral are evolutionary levels.
 - enables understanding and reacting to risks during each iteration along the spiral.
 - Uses:
 - prototyping as a risk reduction mechanism
 - retains the step-wise approach of the waterfall model.

Spiral Model as a Meta Model

Agile Models



What is Agile Software Development?

- **Agile:** Easily moved, light, nimble, active software processes
- **How agility achieved?**
 - Fitting the process to the project
 - Avoidance of things that waste time

Agile Model

- To overcome the shortcomings of the waterfall model of development.
 - Proposed in mid-1990s
- The agile model was primarily designed:
 - To help projects to adapt to change requests
- In the agile model:
 - The requirements are decomposed into many small incremental parts that can be developed over one to four weeks each.

Ideology: Agile Manifesto

- Individuals and interactions *over*
 - process and tools
- Working Software *over*
 - comprehensive documentation
- Customer collaboration *over*
 - contract negotiation
- Responding to change *over*
 - following a plan

<http://www.agilemanifesto.org>

Agile Methodologies

- XP
- Scrum
- Unified process
- Crystal
- DSDM
- Lean

Agile Model: Principal Techniques

- **User stories:**
 - Simpler than use cases.
- **Metaphors:**
 - Based on user stories, developers propose a common vision of what is required.
- **Spike:**
 - Simple program to explore potential solutions.
- **Refactor:**
 - Restructure code without affecting behavior, improve efficiency, structure, etc.

- At a time, only one increment is planned, developed, deployed at the customer site.

Agile Model: Nitty Gritty

- No long-term plans are made.
- An iteration may not add significant functionality,
 - But still a new release is invariably made at the end of each iteration
 - Delivered to the customer for regular use.

Methodology

- **Face-to-face communication favoured over written documents.**
- To facilitate face-to-face communication,
 - Development team to share a single office space.
 - Team size is deliberately kept small (5-9 people)
 - This makes the agile model most suited to the development of small projects.

Effectiveness of Communication Modes

Communication Effectiveness

Cold

Richness of Communication Channel

Hot

Paper

Audiotape

Email conversation

Videotape

Documentation Options

Phone conversation

Video conversation

Face-to-face conversation

Face-to-face at whiteboard

Modeling Options

Copyright 2002-2005 Scott W. Ambler
Original Diagram Copyright 2002 Alistair Cockburn

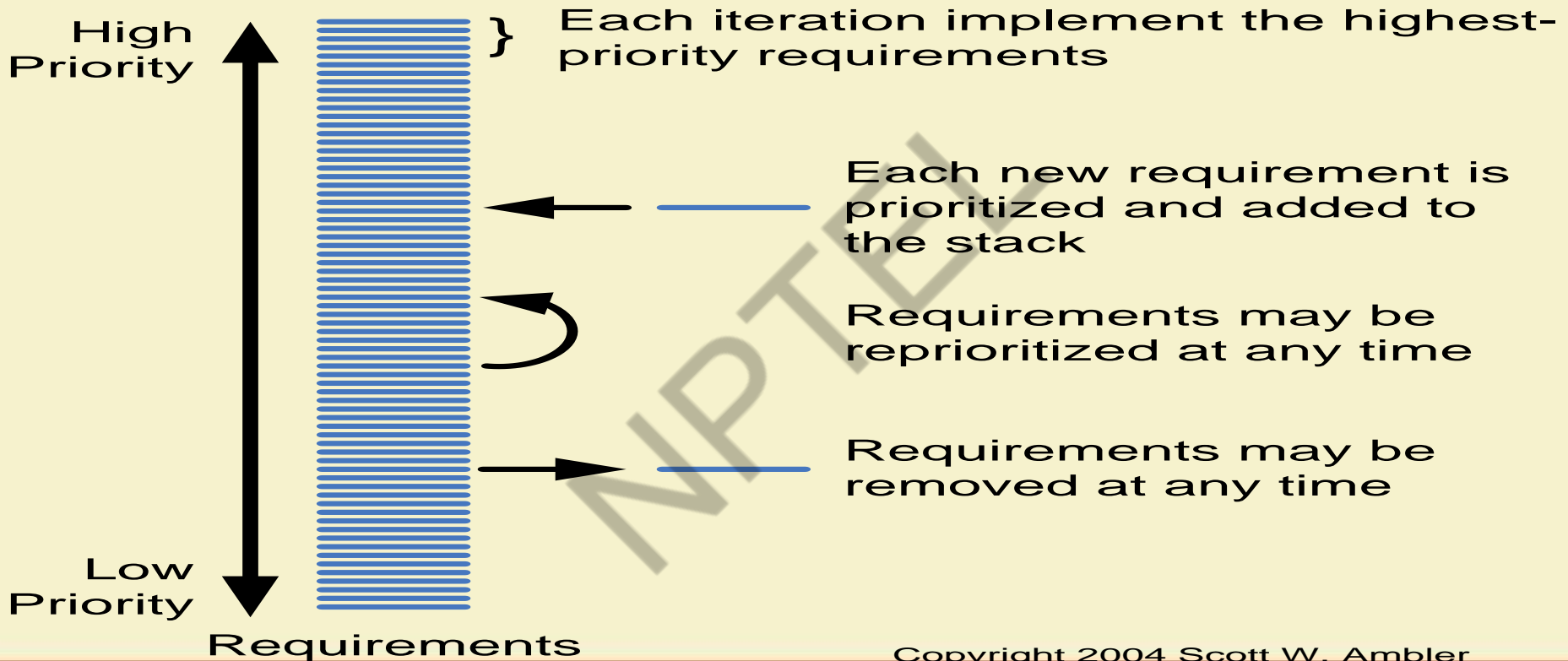
Agile Model: Principles

- The primary measure of progress:
 - **Incremental release of working software**
- Important principles behind agile model:
 - Frequent delivery of versions --- once every few weeks.
 - Requirements change requests are easily accommodated.
 - Close cooperation between customers and developers.
 - Face-to-face communication among team members.

Agile Documentation

- Travel light:
 - You need far less documentation than you think.
- Agile documents:
 - Are concise
 - Describe information that is less likely to change
 - Describe “good things to know”
 - Are sufficiently accurate, consistent, and detailed
- Valid reasons to document:
 - Project stakeholders require it
 - To define a contract model
 - To support communication with an external group
 - To think something through

Agile Software Requirements Management



Copyright 2004 Scott W. Ambler

Adoption Detractors

- Sketchy definitions, make it possible to have
 - Inconsistent and diverse definitions
- High quality people skills required
- Short iterations inhibit long-term perspective
- Higher risks due to feature creep:
 - Harder to manage feature creep and customer expectations
 - Difficult to quantify cost, time, quality.

Agile Model Shortcomings

- Derives agility through developing tacit knowledge within the team, rather than any formal document:
 - Can be misinterpreted...
 - External review difficult to get...
 - When project is complete, and team disperses, maintenance becomes difficult...

Agile Model versus Iterative Waterfall Model

- The waterfall model steps through in a planned sequence:
 - Requirements-capture, analysis, design, coding, and testing .
- Progress is measured in terms of delivered artefacts:
 - Requirement specifications, design documents, test plans, code reviews, etc.
- In contrast agile model sequences:
 - Delivery of working versions of a product in several increments.

Agile Model versus Iterative Waterfall Model

- As regards to similarity:
 - We can say that Agile teams use the waterfall model on a small scale.

Agile versus RAD Model

- Agile model does not recommend developing prototypes:
 - Systematic development of each incremental feature is emphasized.
- In contrast:
 - RAD is based on designing quick-and-dirty prototypes, which are then refined into production quality code.

Agile versus exploratory programming

- Similarity:
 - Frequent re-evaluation of plans,
 - Emphasis on face-to-face communication,
 - Relatively sparse use of documents.
- Agile teams, however, do follow defined and disciplined processes and carry out rigorous designs:
 - This is in contrast to chaotic coding in exploratory programming.

Extreme Programming (XP)



Extreme Programming Model

- Extreme programming (XP) was proposed by Kent Beck in 1999.
- The methodology got its name from the fact that:
 - Recommends taking the best practices to extreme levels.
 - If something is good, why not do it all the time.

Taking Good Practices to Extreme

- **If code review is good:**
 - Always review --- **pair programming**
- **If testing is good:**
 - Continually write and execute test cases --- **test-driven development**
- **If incremental development is good:**
 - Come up with new increments every few days
- **If simplicity is good:**
 - Create the simplest design that will support only the currently required functionality.

Taking to Extreme

- **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)

4 Values

- **Communication:**
 - Enhance communication among team members and with the customers.
- **Simplicity:**
 - Build something simple that will work today rather than something that takes time and yet never used
 - May not pay attention for tomorrow
- **Feedback:**
 - System staying out of users is trouble waiting to happen
- **Courage:**
 - Don't hesitate to discard code

Best Practices

- **Coding:**
 - without code it is not possible to have a working system.
 - Utmost attention needs to be placed on coding.
- **Testing:**
 - Testing is the primary means for developing a fault-free product.
- **Listening:**
 - Careful listening to the customers is essential to develop a good quality product.

Best Practices

- **Designing:**
 - Without proper design, a system implementation becomes too complex
 - The dependencies within the system become too numerous to comprehend.
- **Feedback:**
 - Feedback is important in learning customer requirements.

Extreme Programming Activities

- **XP Planning**

- Begins with the creation of “user stories”
- Agile team assesses each story and assigns a cost
- Stories are grouped to for a deliverable increment
- A commitment is made on delivery date

- **XP Design**

- Follows the KIS principle
- Encourage the use of CRC cards
- For difficult design problems, suggests the creation of “spike solutions”—a design prototype
- Encourages “refactoring”—an iterative refinement of the internal program design

Extreme Programming Activities

- **XP Coding**
 - Recommends the construction of unit test cases *before* coding commences (test-driven development)
 - Encourages “pair programming”
- **XP Testing**
 - All unit tests are executed daily
 - “Acceptance tests” are defined by the customer and executed to assess customer visible functionalities

Full List of XP Practices

1. **Planning** – 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 cycles
3. **Metaphor** – all development is guided by a simple shared story of how the whole system works
4. **Simple design** – system is to be designed as simple as possible
5. **Testing** – programmers continuously write and execute unit tests

Full List of XP Practices

7. **Refactoring** – programmers continuously restructure the system without changing its behavior to remove duplication and simplify
8. **Pair-programming** -- all production code is written with two programmers at one machine
9. **Collective ownership** – anyone can change any code anywhere in the system at any time.
10. **Continuous integration** – integrate and build the system many times a day – every time a task is completed.

Full List of XP Practices

- 11. 40-hour week** – work no more than 40 hours a week as a rule
- 12. On-site customer** – a user is a part of the team and available full-time to answer questions
- 13. Coding standards** – programmers write all code in accordance with rules emphasizing communication through the code

Emphasizes Test-Driven Development (TDD)

- Based on user story develop test cases
- Implement a quick and dirty feature every couple of days:
 - **Get customer feedback**
 - **Alter if necessary**
 - **Refactor**
- Take up next feature

Project Characteristics that Suggest Suitability of Extreme Programming

- Projects involving new technology or research projects.
 - In this case, the requirements change rapidly and unforeseen technical problems need to be resolved.
- Small projects:
 - These are easily developed using extreme programming.

Practice Questions

- What are the stages of iterative waterfall model?
- What are the disadvantages of the iterative waterfall model?
- Why has agile model become so popular?
- What difficulties might be faced if no life cycle model is followed for a certain large project?

Suggest Suitable Life Cycle Model

- A software for an academic institution to automate its:
 - Course registration and grading
 - Fee collection
 - Staff salary
 - Purchase and store inventory
- The software would be developed by tailoring a similar software that was developed for another educational institution:
 - 70% reuse
 - 10% new code and 20% modification

Practice Questions

- Which types of risks can be better handled using the spiral model compared to the prototyping model?
- Which type of process model is suitable for the following projects:
 - A customization software
 - A payroll software for contract employees that would be add on to an existing payroll software

Practice Questions

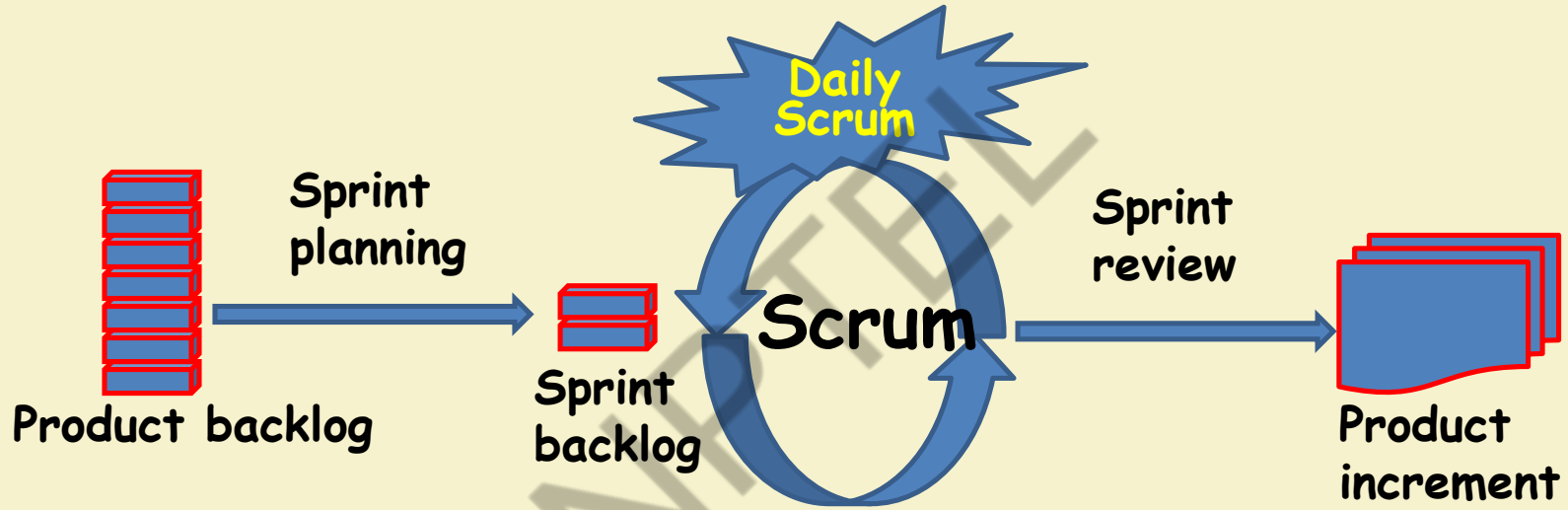
- Which lifecycle model would you select for the following project which has been awarded to us by a mobile phone vendor:
 - A new mobile operating system by upgrading the existing operating system
 - Needs to work well efficiently with 4G systems
 - Power usage minimization
 - Directly upload backup data on a cloud infrastructure maintained by the mobile phone vendor

Scrum



Scrum: Characteristics

- Self-organizing teams
- Product progresses in a series of month-long **sprints**
- Requirements are captured as items in a list of **product backlog**
- One of the agile processes



Sprint

- Scrum projects progress in a series of “sprints”
 - Analogous to XP iterations or time boxes
 - Target duration is one month
- **Software increment is designed, coded, and tested during the sprint**
- **No changes entertained during a sprint**

Scrum Framework

- **Roles** : Product Owner, ScrumMaster, Team
- **Ceremonies** : Sprint Planning, Sprint Review, Sprint Retrospective, and Daily Scrum Meeting
- **Artifacts** : Product Backlog, Sprint Backlog, and Burndown Chart

Key Roles and Responsibilities in Scrum Process

- **Product Owner**
 - Acts on behalf of customers to represent their interests.
- **Development Team**
 - Team of five-nine people with cross-functional skill sets.
- **Scrum Master (aka Project Manager)**
 - Facilitates scrum process and resolves impediments at the team and organization level by acting as a buffer between the team and outside interference.

Product Owner

- Defines the features of the product
- Decide on release date and content
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results.

The Scrum Master

- Represents management to the project
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

Scrum Team

- Typically 5-10 people
- Cross-functional
 - QA, Programmers, UI Designers, etc.
- Teams are self-organizing
- Membership can change only between sprints

Ceremonies

- Sprint Planning Meeting
- Sprint
- Daily Scrum
- Sprint Review Meeting

Sprint Planning

- Goal is to produce Sprint Backlog
- Product owner works with the Team to negotiate what Backlog Items the Team will work on in order to meet Release Goals
- Scrum Master ensures Team agrees to realistic goals

Sprint

- Fundamental process flow of Scrum
- A month-long iteration, during which an incremental product functionality completed
- NO outside influence can interfere with the Scrum team during the Sprint
- Each day begins with the Daily Scrum Meeting

- Daily
- 15-minutes
- Stand-up meeting
- Not for problem solving
- Three questions:
 1. What did you do yesterday
 2. What will you do today?
 3. What obstacles are in your way?

Daily Scrum

Daily Scrum

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the Team

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features
- Informal
 - 2-hour prep time rule
- Participants
 - Customers
 - Management
 - Product Owner
 - Other engineers

Sprint Review Meeting

Product Backlog

- A list of all desired work on the project
 - Usually a combination of
 - story-based work (“let user search and replace”)
 - task-based work (“improve exception handling”)
- List is prioritized by the Product Owner
 - Typically a Product Manager, Marketing, Internal Customer, etc.

Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
 - Managed and owned by Product Owner
 - Spreadsheet (typically)
 - Usually is created during the Sprint Planning Meeting

Sample Product Backlog

	Item #	Description	Est	By
Very High				
	1	Finish database versioning	16	KH
	2	Get rid of unneeded shared Java in database	8	KH
		Add licensing	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
		Enforce unique names	-	-
	7	In main application	24	KH
	8	In import	24	AM
		Admin Program	-	-
	9	Delete users	4	JM
		Analysis Manager	-	-
		When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	10	Query	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
		Population Genetics	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	Add icons for v1.1 or 2.0	-	-
		Pedigree Manager	-	-
	20	Validate Derived kindred	4	KH
Medium				
		Explorer	-	-
		Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

Sprint Backlog

- A subset of Product Backlog Items, which define the work for a Sprint
 - Created by Team members
 - Each Item has it's own status
 - Updated daily

Sprint Backlog during the Sprint

- Changes occur:
 - Team adds new tasks whenever they need to in order to meet the Sprint Goal
 - Team can remove unnecessary tasks
 - But: Sprint Backlog can only be updated by the team
- Estimates are updated whenever there's new information

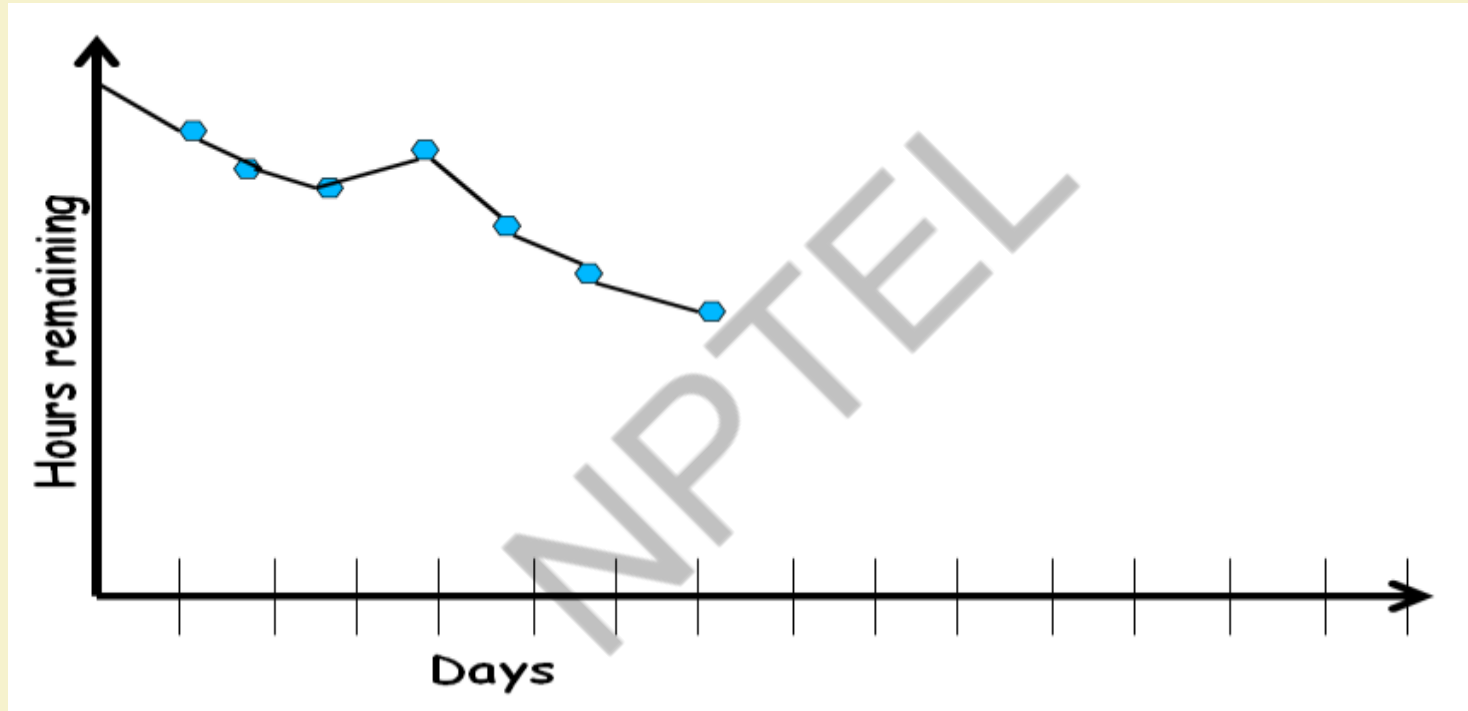
Burn down Charts

- Are used to represent “work done”.
- Are remarkably simple but effective Information disseminators
- 3 Types:
 - Sprint Burn down Chart (progress of the Sprint)
 - Release Burn down Chart (progress of release)
 - Product Burn down chart (progress of the Product)

Sprint Burn down Chart

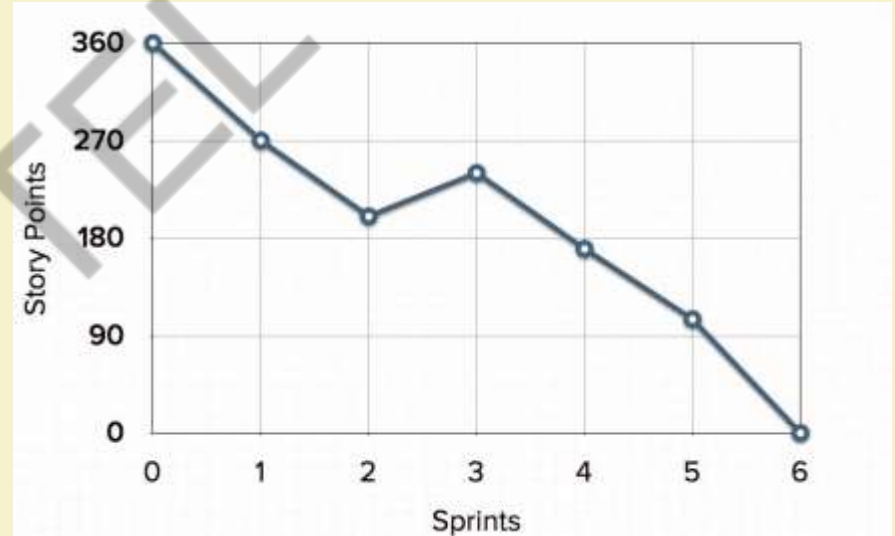
- Depicts the total Sprint Backlog hours remaining per day
- Shows the estimated amount of time to release
- Ideally should burn down to zero to the end of the Sprint
- Actually is not a straight line

Sprint Burndown Chart



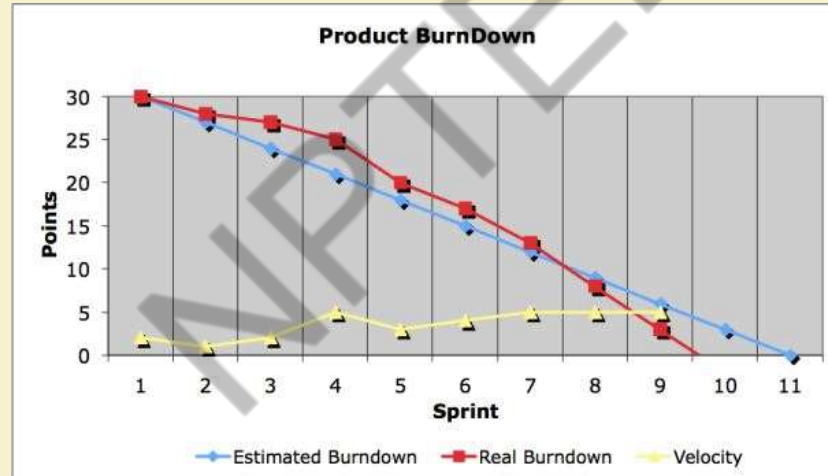
Release Burndown Chart

- Will the final release be done on right time?
- How many more sprints?
- X-axis: sprints
- Y-axis: amount of story points remaining



Product Burndown Chart

- Is a “big picture” view of project’s progress (all the releases)



Scalability of Scrum

- A typical Scrum team is 6-10 people
- Jeff Sutherland - up to over 800 people
- "Scrum of Scrums" or what called "Meta-Scrum"
- Frequency of meetings is based on the degree of coupling between packets

Thank You!!



Agile vs. Plan-Driven Processes

- | | |
|--|--|
| 1. Small products and teams: <ul style="list-style-type: none">● Scalability limited | 1. Large products and teams; <ul style="list-style-type: none">● hard to scale down |
| 2. Untested on safety-critical products | 2. Proven for highly critical products; |
| 3. Good for dynamic, but expensive for stable environments. | 3. Good for stable: <ul style="list-style-type: none">● But expensive for dynamic environments |
| 4. Require experienced personnel throughout | 4. Require only few experienced personnel |
| 5. Personnel thrive on freedom and chaos | 5. Personnel thrive on structure and order |