Injibara University

College of Engineering and Technology

Department of Software Engineering

Course Title: Software Engineering

Course Code:CoSc3061

Molla K. (MSc)

Software Engineering

Chapter -one Introduction

Definition of Software Eengineering (SWE)

- It is an *engineering discipline* concerned with *all aspects of SW production* starting from the early stages of system specification through to the maintenance of the system after it has started to be used.
  - It is an approach to develop software(to building construction).
  - It is a methodology that includes process methods tools and techniques for the manufacturing of software product which is:
  - Timely produced, Cost effective, User-friendly, Portable, Reliable , maintainable and reusable.

Cont'd

Engineering Discipline:

- ✓İmplies solving a well-defined (or in SW engineering vaguely defined) problem optimally using resources (e.g., time, man power and machine power) and remaining within
  - ✓Organizational (I.E., The Customer)
  - ✓Financial, and Other Possible Constraints.
- ✓ *Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods .*
- ✓ Software engineering is a systematic approach for the development, operation and maintenance of computer software systems.
- ✓ It is the process of designing and building something that serves as a particular purpose.

Cont’d

- ➢Software engineering is a ***modeling*** activity.
- ➢Software engineers deal with complexity through modeling, by focusing at any one time on only the relevant details and ignoring everything else.
- ✓Software engineering is a ***problem-solving*** activity.
- Models are used to search for an acceptable solution.
- Software engineers do not have **infinite** resources and are constrained by **budget** and **deadlines**.
- ✓It is the combination of all the tools, techniques, and processes that are used in software production.

Cont'd

- ❖ Software engineering is a ***knowledge acquisition*** activity.
- ❖ In modeling the application and solution domain, software engineers collect data, organize it into information, and formalize it into knowledge.
- ▪ Software engineering is a ***rationale-driven*** ***activity***.
- ▪ When acquiring knowledge and making decisions about the system or its application domain, software engineers also need to capture the context in which decisions were made and the rationale behind these decisions.
- ▪ **The process of developing a software product using software engineering principles and methods is referred to as Software Evolution .**

Cont'd

- It can be defined as the construction of *quality* software with *a limited budget* and *a given deadline* in the *context of constant change.*
- Software engineering (SE) is an intellectual activity and thus human-intensive.
- Software is built to meet a certain functional goal and satisfy certain qualities.
- Software processes also must meet certain qualities.
- ***It is developed by Ivar Jacbson in 1992.***
- ***It is the first OO methodology.***
- ➢ ***The use of use case in design is started by OOSE technique i.e introduce OO Methdology.***
- ***It includes requirments, analysis, design and implmentation and testing.***

Software Engineering Concepts: Definitions

- *Project* – set of activities to develop a software system.
- *Activity* – a **phase** in which related tasks are carried out.
- *Task* – effort that uses resources and produces work Product.
- *Resources*– are assets that are used to accomplish work. time, equipment, people (participants).
- *Work Product* – a model, system, or artifact.

Analysis versus Design

- An analysis technique helps elaborate the customer requirements through careful thinking:
  - And at the same time consciously avoids making any decisions regarding implementation.
- The design model is obtained from the analysis model through transformations over a series of steps:
  - Decisions regarding implementation are consciously made.

TWO ORTHOGONAL VIEWS OF SOFTWARE

- Traditional technique
- Object oriented methodologies

TRADITIONAL APPROACH	OBJECT ORIENTED SYSTEM DEVELOPMENT
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step of process	Focuses on object, classes, modules that can be easily replaced, modified and reused.
Moving from one phase to another phase is complex.	Moving from one phase to another phase is easier.
Increases duration of project	decreases duration of project
Increases complexity	Reduces complexity and redundancy

Object Oriented Systems Development Methodology

- Object Oriented systems development methodology develops software by building objects that can be easily replaced , modified and reused.
- It is a system of cooperative and collaborating objects.
- Each objects has attributes (data) and methods (functions).
- OO modeling is entirely a new way of thinking about problems.

Con...

- This methodology is all about visualizing the things by using models organized around real world concepts.
- OO models help in understanding problems, communicating with experts from a distance, modeling enterprises, and designing programs and database.
- In OOM, the modeling passes through the following processes:
  - System Analysis
  - System Design
  - Object Design, and
  - Final Implementation.

Con...

- **System Analysis**: In this stage a statement of the problem is formulated and a model is build by the analyst in encouraging real-world situation.

- This phase show the important properties associated with the situation.

- **System Design**: At this stage, the complete system architecture is designed.

- This is the stage where the whole system is divided into subsystems, based on both the system analysis model and the proposed architecture of the system.

con...

- **Object Design**: At this stage, a design model is developed based on the analysis model which is already developed in the earlier phase of development.
- The object design decides the data structures and algorithms needed to implement each of the classes in the system with the help of implementation details given in the analysis model.
- **Final Implementation:** At this stage, the final implementation of classes and relationships developed during object design takes place a particular programming language, database, or hardware implementation (if needed).

con...

- The whole object oriented modeling is covered by using three kinds of models for a system description. These models are:
- **Object Model** :are used for describing the objects in the system and their relationship among each other in the system.
- **Dynamic Model** :describes interaction among objects and information flow in the system.
- **Functional Model** :The data transformations in the system are described by a functional model.

Why an Object Orientation?

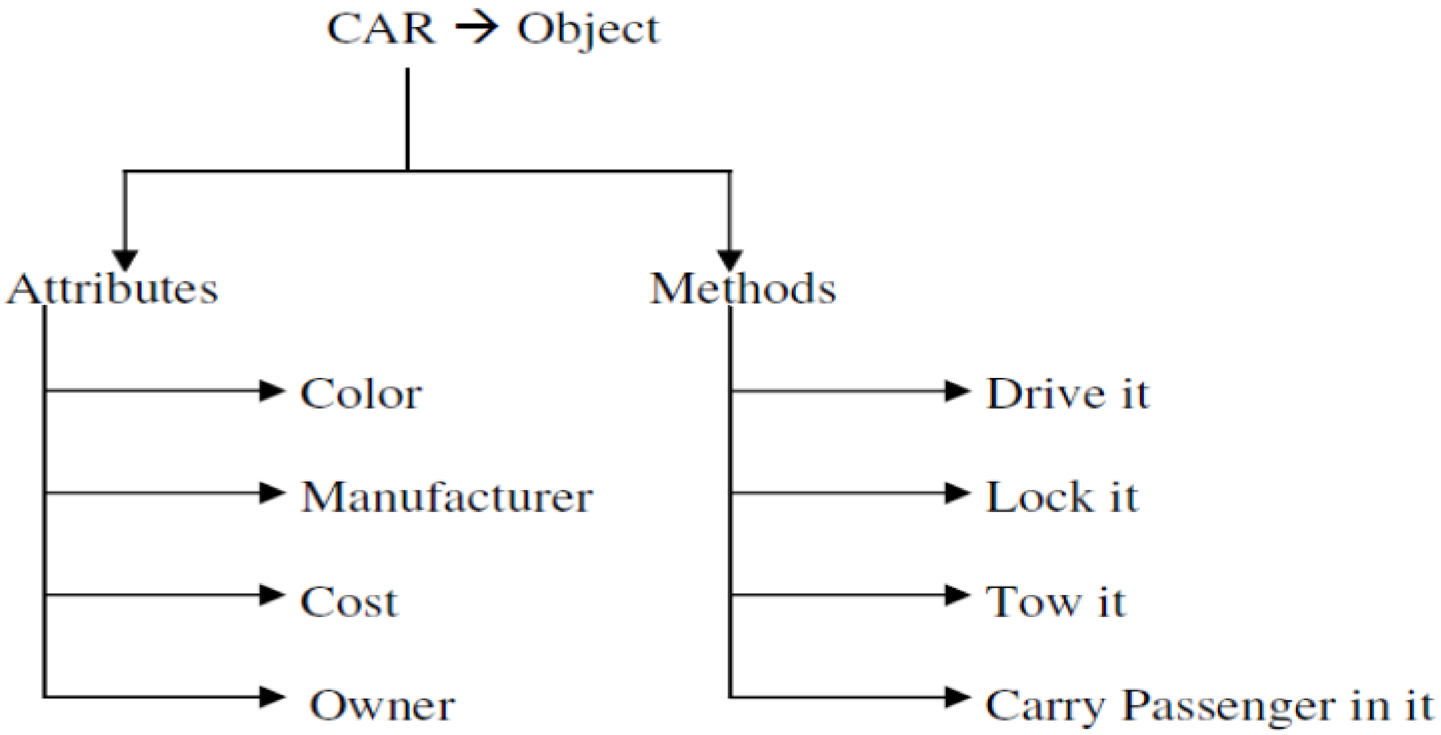
- Object oriented systems are
  - Easier to adapt to changes
  - More robust
  - Easier to maintain
  - Promote greater design and code reuse
  - Creates modules of functionality

Object

- An object is a real-world element in an object-oriented environment that may have a physical or a conceptual existence. Each object has:
- **Identity** that distinguishes it from other objects in the system.
- **State** that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- **Behavior** that represents externally visible activities performed by an object in terms of changes in its state.

Con…

- A car is an object a real-world entity, identifiably separate from its surroundings.
- A car has a well-defined set of attributes in relation to other object.



con…

- Company ABC needs to implement an order-entry system that takes orders from customers for a variety of products.
- What are the objects in this problem domain?
- Hint: Objects are usually described as nouns.

Customer

Order

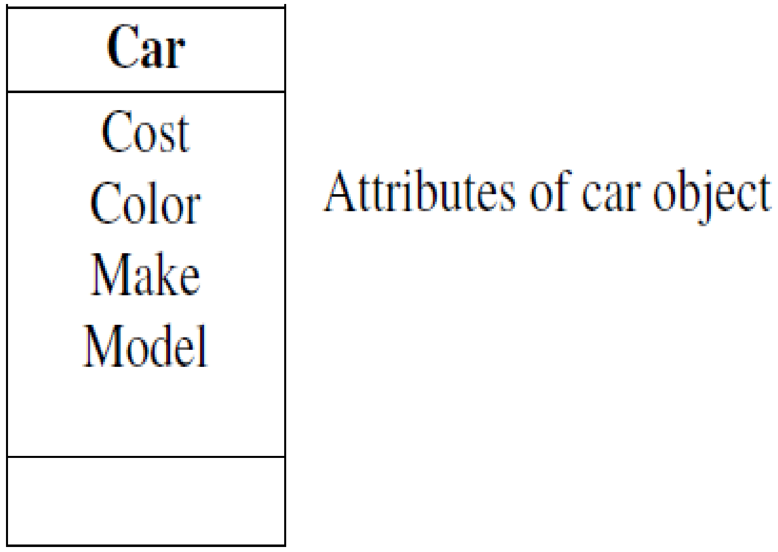
Product

Attributes Object State and Properties.

- **Attributes**: It is a simple piece of data used to represent the properties of an object.
  - ➢ Data of an object.
  - ➢ Properties of an object. e.g name, adress
- **Methods**:
  - ✓ Procedures of an object.or
  - ✓Behaviour of an object.

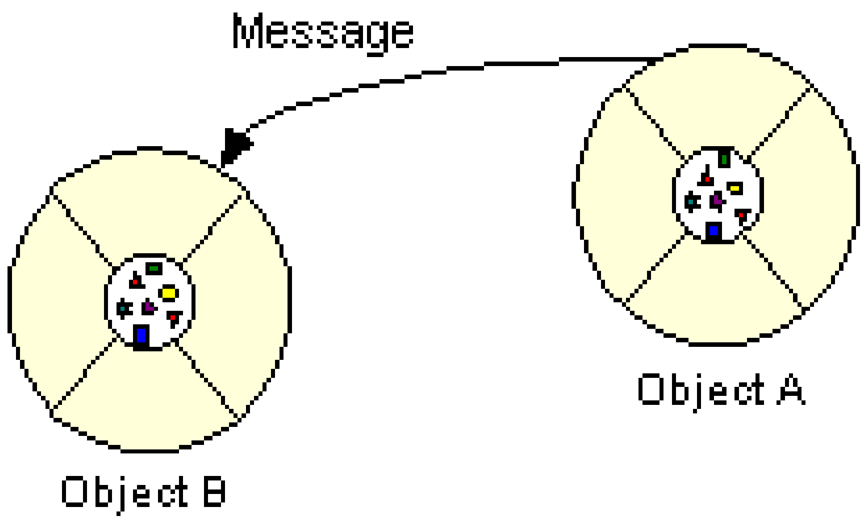
Con…

- **Properties** represent the **state** of an object.
- In an object oriented methods we want to refer to the description of these properties rather than how they are represented in a particular programming language.



Con...

- Objects interact and communicate with each other by sending messages to each other.
- When object A wants object B to perform one of B's methods, object A sends a message to object B.



Class

- Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields and properties) and the thing's behaviors (the things it can do, or methods, operations or features).
- A class is a ***template*** used to create *multiple objects* with similar feature.

Encapsulation

- In object-oriented programming, objects interact with each other by messages.
- The only thing that an object knows about another object is the object's interface.
- Each object's data and operation is **hidden** from other objects.
- In other words, the interface encapsulate the object's code and data.
- This allows the developer to separate an object's implementation from its behavior.

Con…

- For example, if we have two objects, Object A and Object B, if A sends a message to B, it does not matter to the user how the developer implemented the code to handle this message.
- All the sending object needs is the correct protocol for interacting with the object B.
- The developer can change the implementation at any time, but the message would still work because the interface is the same.

Inheritance

- Another important concept of object-oriented programming is inheritance.
- Inheritance allows a class to have the same behavior as another class and extend or tailor that behavior to provide special action for specific needs.
- Classes that inherit from a class are called **subclasses**.
- The class a subclass inherits from are called **super class**.

Con. . .

- Let's use the following application as an example. Both Graduate class and Undergraduate class have similar behavior such as managing a name, an address, a major, and a GPA. Rather than put this behavior in both of these classes, the behavior is placed in a new class called Student. Both Graduate and Undergraduate become subclass of the Student class, and both inherit the Student behavior.
- Both Graduate and Undergraduate classes can then add additional behavior that is unique to them. For example, Graduate can be either Master's program or PhD program. On the other hand, Undergraduate class might want to keep track of either the student is Freshman, Sophomore, Junior or Senior.

Abstraction

- Abstraction is simplifying complex reality by modeling classes appropriate to the problem, and working at the most appropriate level of inheritance for a given aspect of the problem.

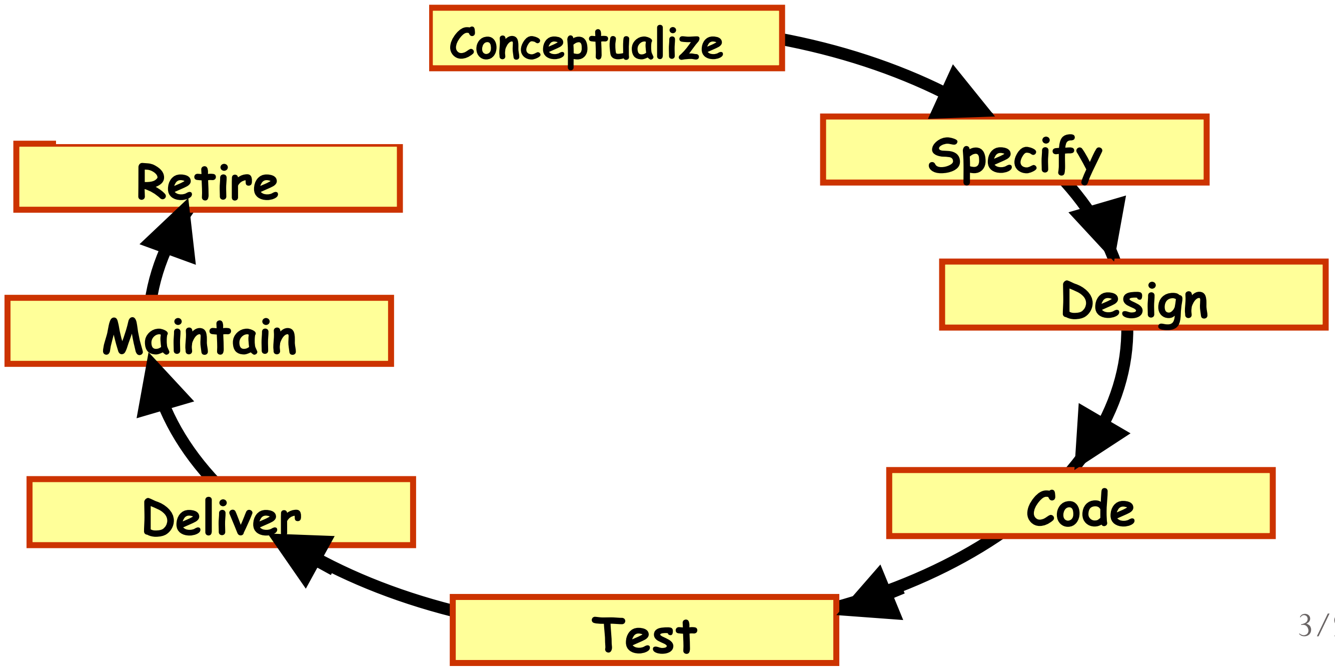
Unified Approach (UA): -

- UA is a methodology for software development is based on methodologies by **Booch**, **Rumbaugh** **and** **Jacobson**.
- It tries to combine best practices, processes, and guidelines along with UML notations & diagrams for better understanding OO concepts and system development.
- UA to s/w development revolves around the following process & concepts - Use-case driven development, OO analysis, OO design, Incremental development and prototyping and Continuous testing.
- The *diagrammatic representation of processes & components of UA.*

Software processes & Models

Software Life Cycle

- Breaks software project activities into tasks and chunks of tasks into phases, orders them.
- A software process is a set of related activities that leads to the production of a software product.
- Software process model is a simplified representation of a software process.



Life Cycle Model

- **A software life cycle model (also called process model and at times 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
- The project manager must identify a suitable life cycle model.
  - Make the project team to adhere/remain to it.
  - Primary advantage of following 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.
  - Code → Test → Design
  - Code → Design → Test → Code
  - Specify → code → Design → Test etc.

Life Cycle Model (CONT.)

- A software project will rarely succeed if:
  - One engineer starts writing code,
  - Another concentrates on writing the test document first,
  - Yet another engineer first defines the file structure.
  - Another focuses on design.

Why Model Life Cycle?

- A documented model:
  - Helps common understanding of activities among the software developers.
  - Helps to identify inconsistencies, redundancies, and errors in the development process.
  - Helps in tailoring/ modifying a process model for specific projects.

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.
- **A phase can start**:
  - Only if its phase-entry criteria have been satisfied.
- **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.**
  - **How an SRS should be tested? What should be tested in an SRS?**

Milestone

- Setting milestones makes it easier to manage software projects:
  - Helps to follow the progress of the project accurately.
  - Phase entry and exit are important milestones.

Life Cycle and Project Management

- When a life cycle model is followed project management is easier:
  - For example: the project manager can at any time fairly accurately tell,
    - At which stage (e.g., Design, code, test, etc.) of the project is.
    - How much time would it take to complete.

Project Management Without a Life Cycle Model…

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

Life Cycle Model (CONT.)

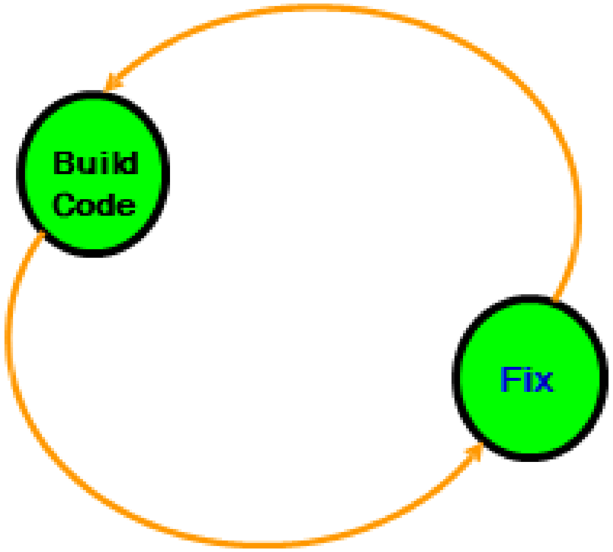
- Many life cycle models have been proposed.
- We confine our attention to a few important and commonly used models.
  - √ Build & Fix
  - √ Waterfall
  - √ V- model,
  - √ Evolutionary,
  - √ Prototyping
  - √ Spiral model, and

  $\left.\begin{array}{l}\\ \\ \\ \\ \\ \end{array}\right\}$ **Traditional models**

  - ❖ Agile models

Build & Fix Model

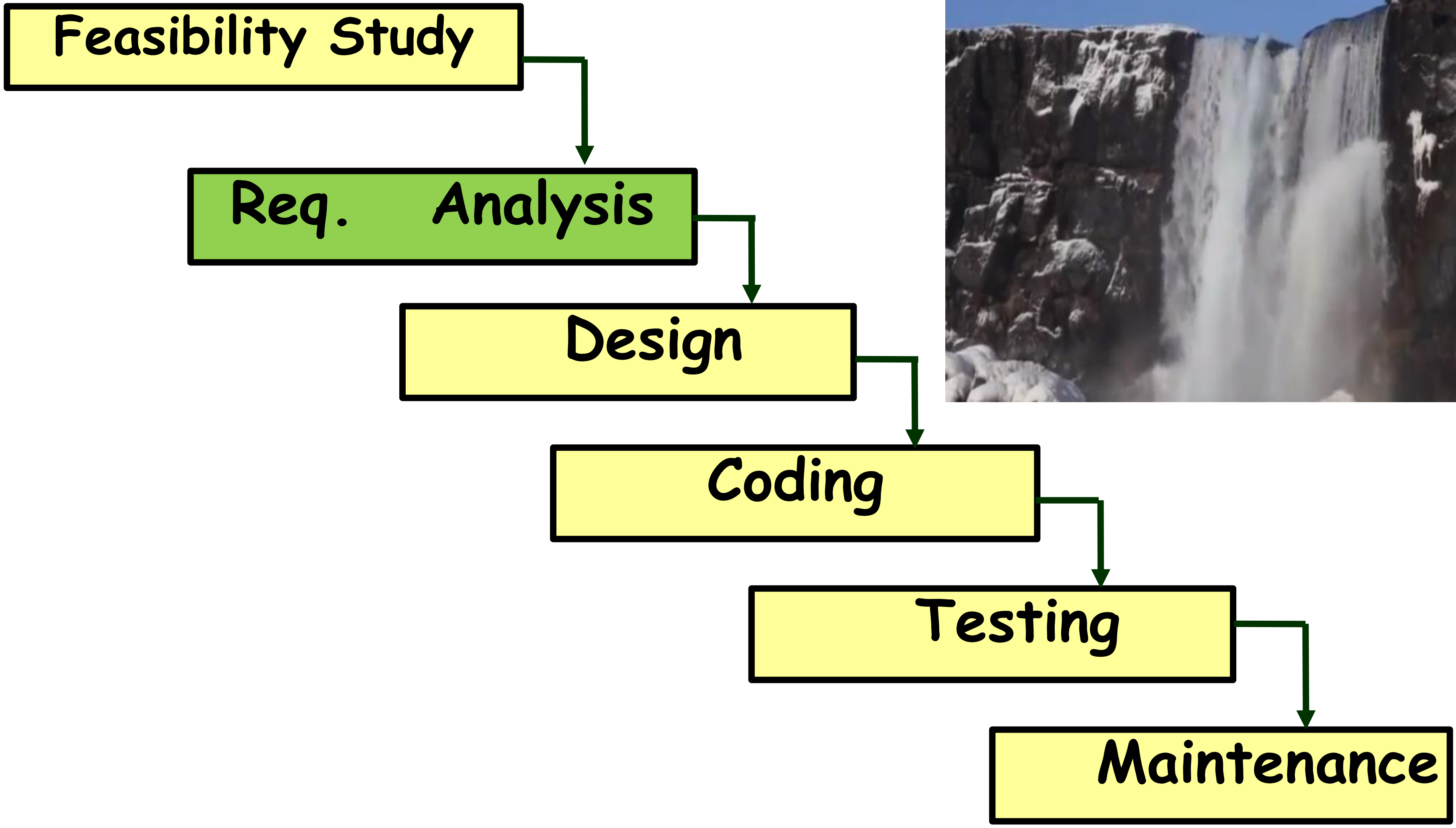
- ❖ Product is constructed without specifications or any attempt at design
- ❖ Adhoc approach and not well defined
- ❖ Simple two phase model
- ❖ Suitable for small programming exercises of **100** or **200** lines
- ❖ Requires less project planning.
- ❖ Unsatisfactory for software for any reasonable size.
- ❖ Maintenance is practically not possible.
- ❖ Requires less experience to execute or manage other than the ability to program.



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.
- The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

Classical Waterfall Model



Feasibility Study

- A **Feasibility** study is a study, usually done by engineers, that establishes whether conditions are right to implement a particular project.
- **Feasibility** studies can be done for many purposes, and are sometimes done in IT in order to look at feasibility for new hardware and software setups.
- **Feasibility** Study in Software Engineering is a study to evaluate feasibility of proposed project or system.
- It is carried out based on many purposes to analyse whether software product will be right in terms of development, establishment, contribution of project to the organization.

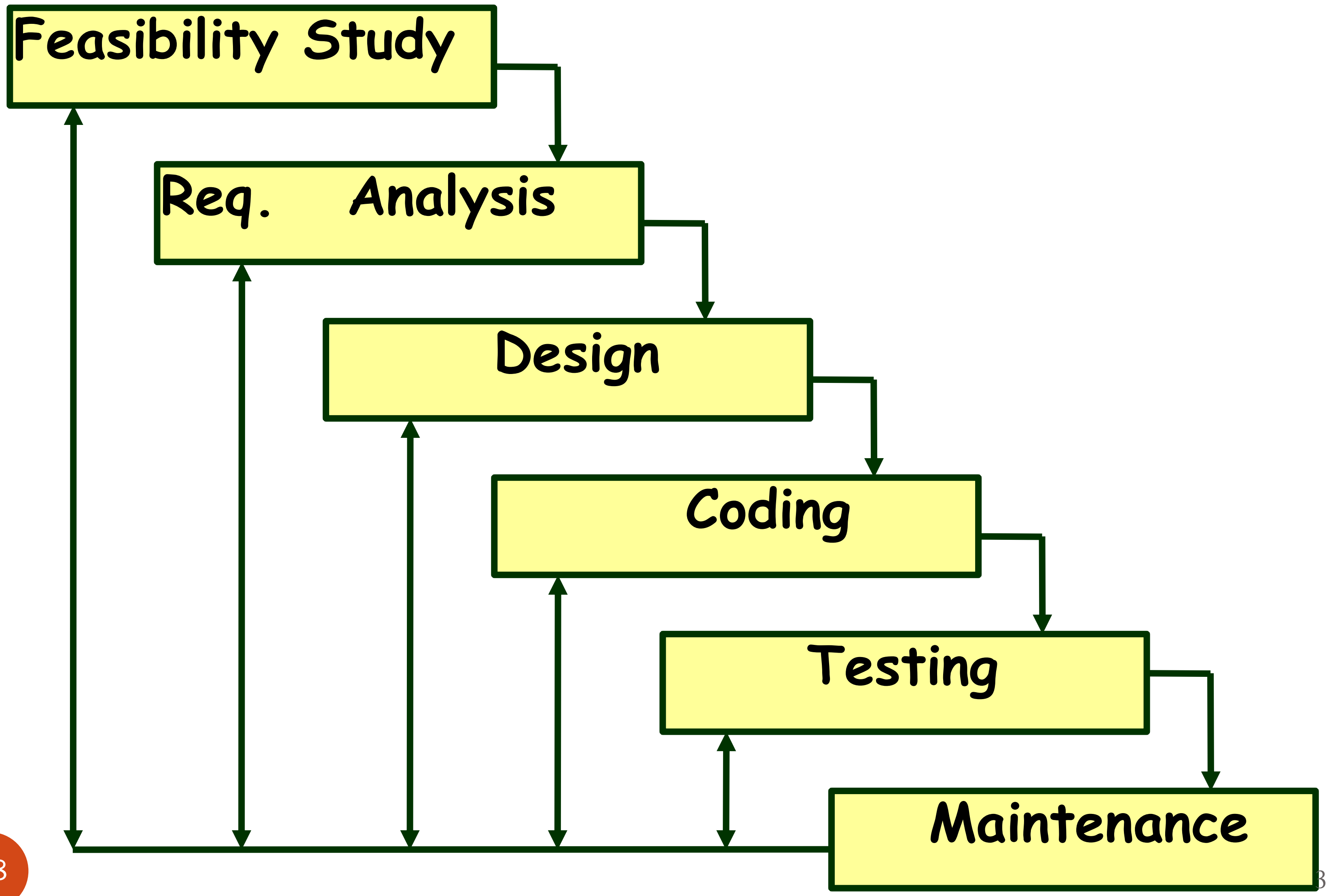
Iterative Waterfall Model

- Classical waterfall model is idealistic one:
- Since it assumes that no development error is always committed/ assign by the engineers during any of the life cycle phases.
  - Assumes that no defect is introduced during any development activity.
  - In practice developers commit errors:
- Defects do get introduced in almost every phase of the life cycle.
- Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase

(CONT.)

- Once a defect is detected:
  - The phase in which it occurred and parts of the work already completed following phases needs to be reworked.
- Therefore need feedback paths in the classical waterfall model.

Iterative Waterfall Model (CONT.)



Waterfall Strengths

- Easy & simple to understand, easy to use.
- Provides a reference to inexperienced staff.
- Milestones are well understood by the team.
- Provides requirements stability.
- Facilitates strong management control (plan, staff, track).
- Easy to manage.
- Phases are processed & completed one at time.
- Requirements are very well understood.
- Clearly defined stages .

Waterfall Deficiencies

- All requirements must be known upfront.
- Can give a false idea of progress.
- Integration is one big bang at the end.
- Little opportunity for customer to pre-view the system.
- No working s/w is produced until late during the life cycle.
- High amount of risk .
- Not good for OOP.
- Poor for a large and on going projects.

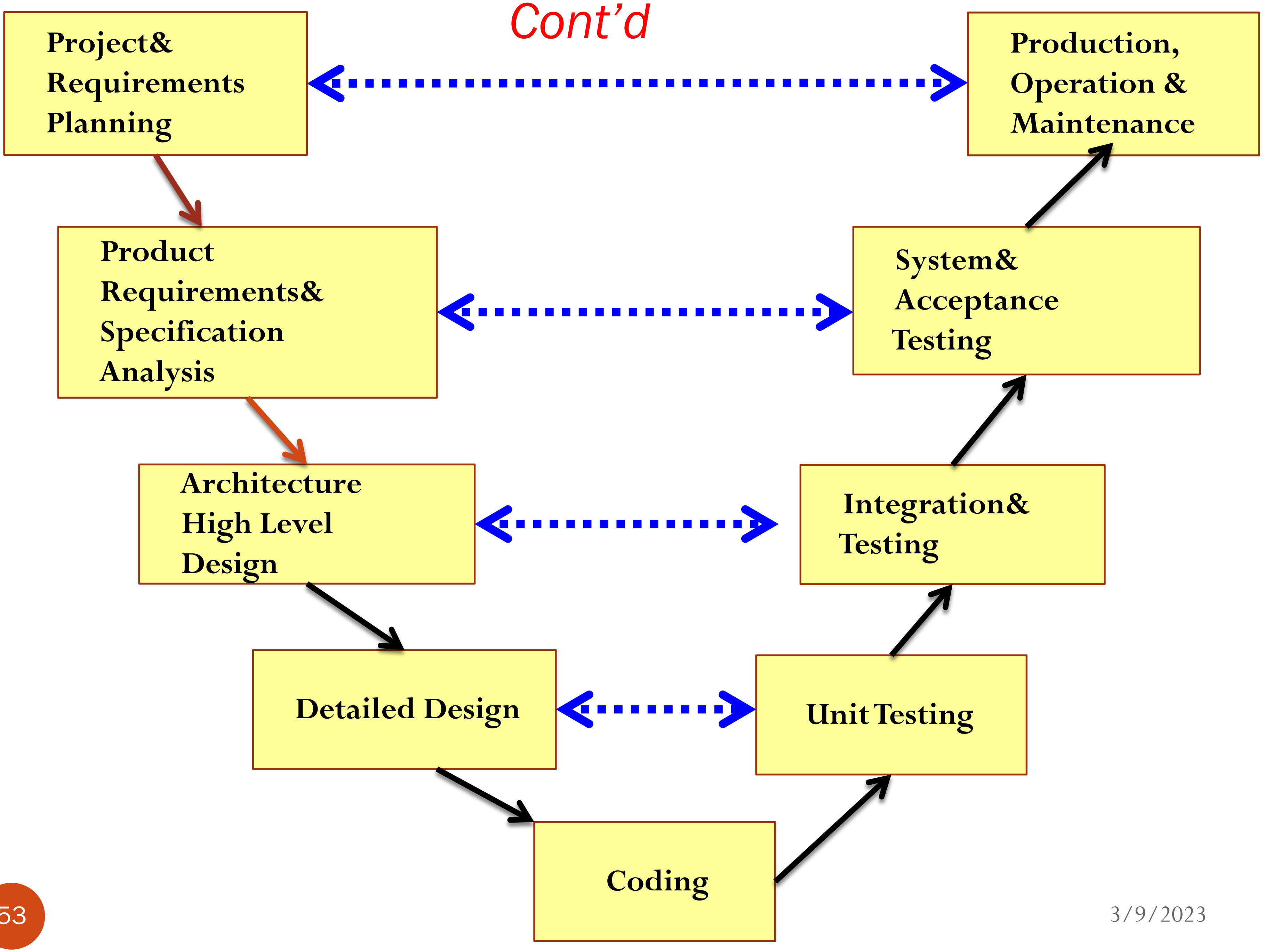
When to use the Waterfall Model?

- Requirements are well known and stable/constant.
- Technology is understood.
- Experienced Development team.

V-Model

- It is a alternative 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.

Cont'd


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

V Model: Strengths

- Starting from early phases of software development:
  - Emphasize 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 handle later changes in requirements.
- Does not support risk analysis activities.

When to use V Model

- Natural choice for systems requiring high reliability/consistency:
  - Embedded control applications
- All requirements are known up-front.
- Solution and technology are known.

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, consistency,
  - Inefficient performance.

Reasons for prototyping

- Learning by doing: useful where requirements are only partially known.
- Improved communication.
- Improved user involvement.
- Reduces the need for documentation
- Reduces maintenance costs i.e. Changes after the application goes live

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.

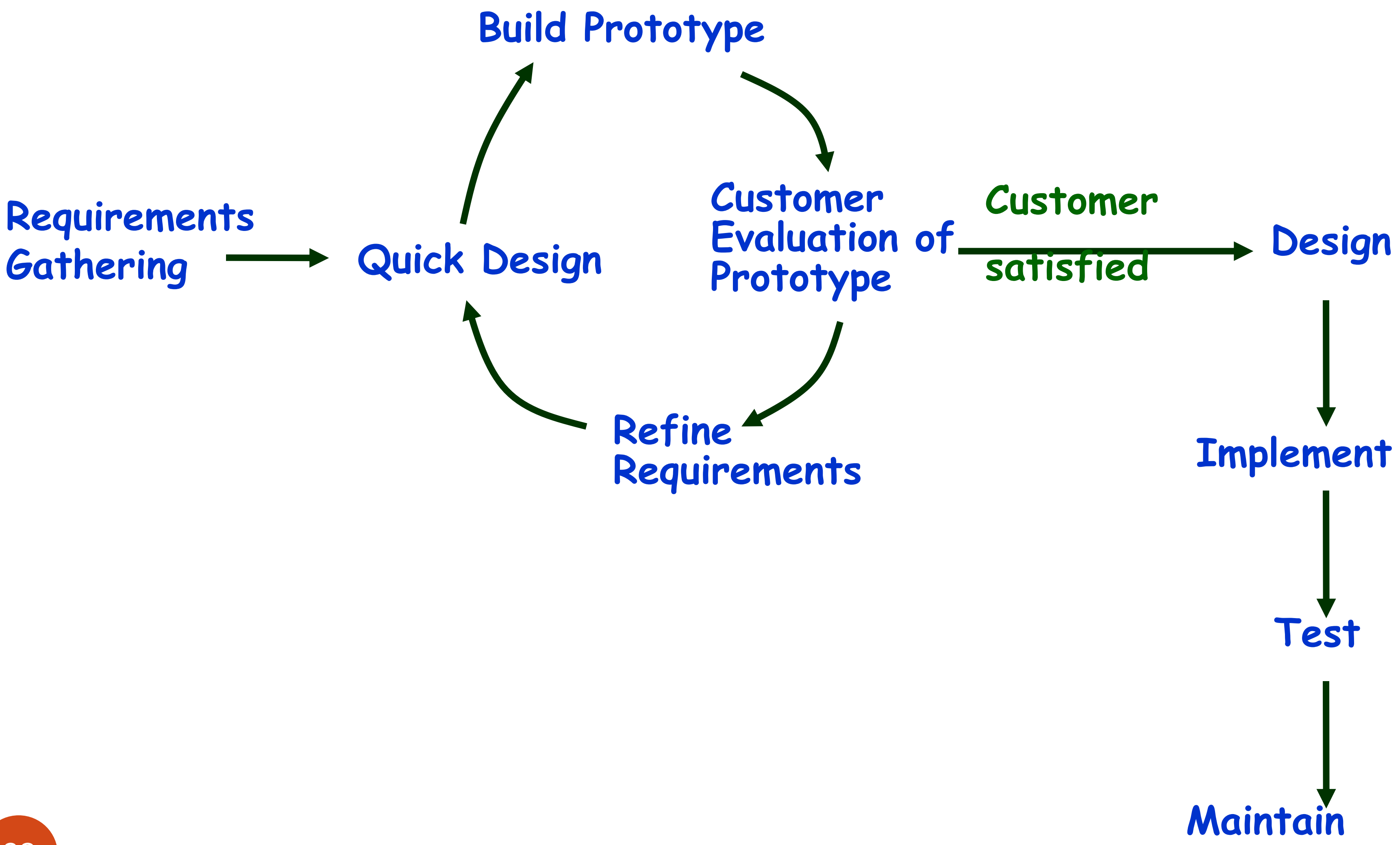
Prototyping Model (CONT.)

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

Prototyping Model **(CONT.)**

- The developed prototype is submitted to the customer for his evaluation:
  - Based on the user feedback, the prototype is refined/improved.
  - This cycle continues until the user approves the prototype.
- The actual system is developed using the waterfall approach.

Prototyping Model (CONT.)



Prototyping: Advantages

- The resulting software is more usable.
- User needs are better accommodated/contained.
- The design is of higher quality.
- The resulting software is easier to maintain.
- **In prototype model reqmts change is allowed.**
- In general, the development gains less effort.

Prototyping: Disadvantages

- Sometimes expensive.
- Susceptible/ Vulnerable to over-engineering:
  - Designers start to incorporate sophistications that they could not integrate in the prototype.

When to use prototype Model

- √ Prototype model should be used when the desired system needs to have a lot of interaction with the end user.
- √ Unclear requirement: when requirements are not properly understand.
- √ Complicated and large system.
- √ Design of user interface.
- √ Unclear solution.

Evolutionary Model

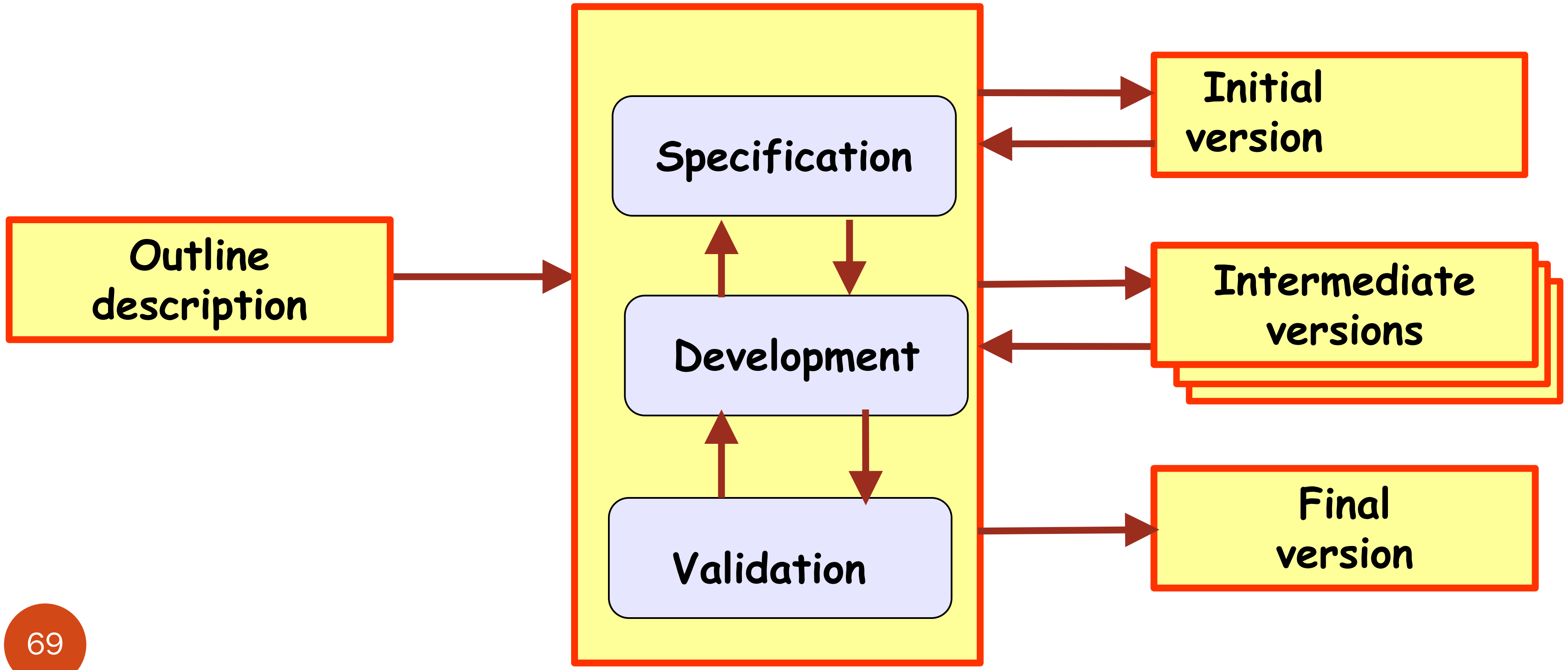
- It is called successive version or incremental model.
- Here the s/w is first broken down into several functional unit w/h can incrementally constructed and delivered.
- Irregular requirements specification.
- Identify the core and other parts to be developed incrementally.
- Develop the core parts using iterative waterfall model.
- Collect customer feedback and modify reqmts.
- Develop the next identified feature using iterative water fall model.
- After all feature are complete, maintenance goes on.
  - First develop the core modules of the software.
  - The initial skeletal software is developed into increasing levels of capability.
    - By adding new functionalities in successive versions.

Evolutionary Model (CONT.)

- Successive version of the product:
  - functioning systems able of performing some useful work.
  - A new release may include new functionality.
    - Also existing functionality in the current release might have been enhanced.
    - It has the same phases with waterfall model but they are applied in cyclical manner.

Evolutionary Model (CONT.)

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



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 all development is complete.

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:
  - Much before fully working system is developed.
- Core modules get tested carefully:
  - Reduces chances of errors in final product.
- Customers involved in the process:
  - Therefore more likely to meet the user requirement.
- Early and frequent testing:
  - More likely to identify problems.

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 poorly structured:
  - Continual/repeated , unpredictable changes tend to reduce the software structure.
- Systems may not even converge or join to a final version.

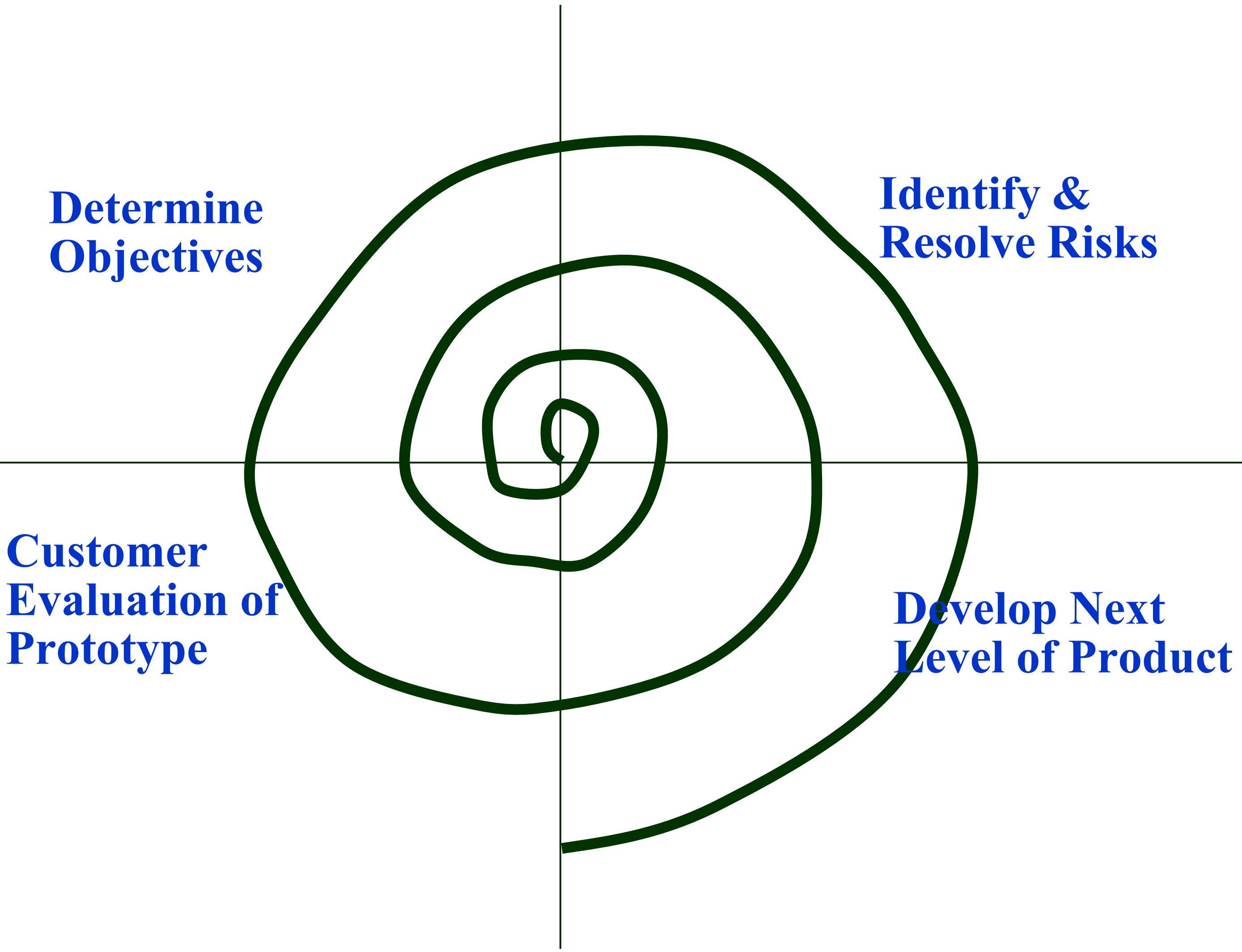
Spiral Model

- Proposed by Boehm in 1988.
- Each loop of the spiral model 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.
- **It is the combination of both waterfall and iterative model.**

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 (CONT.)


Determine Objectives
Identify & Resolve Risks
Customer Evaluation of Prototype
Develop Next Level of Product

Objective Setting (First Quadrant)

- Identify objectives of the phase,
- Examine the risks associated with these objectives.
  - Risk:
    - Any difficult condition that might slow down successful completion of a software project.
- Find alternate solutions possible.

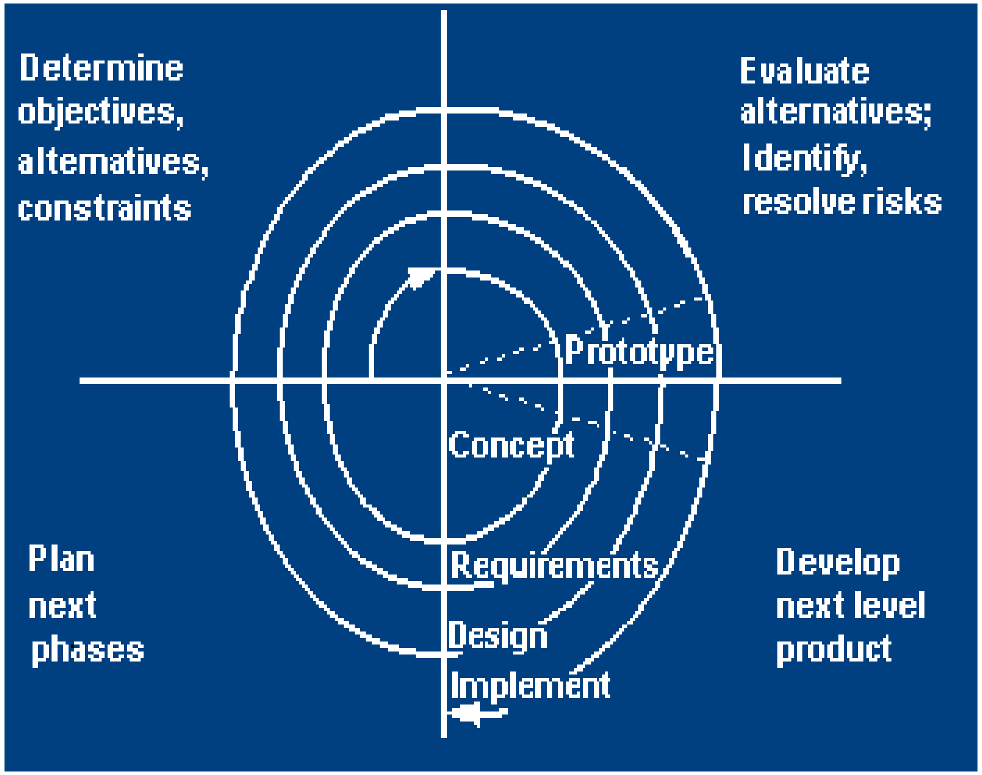
Risk Assessment and Reduction (Second Quadrant)

- 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 the 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.

Spiral SDLC Model


Determine objectives, alternatives, 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

- 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

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 Quadrant

Develop next-level product

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

Spiral Quadrant
Plan next phase

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

Spiral Model strength

- Provides early indication of impossible 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 joined to all lifecycle steps.
- Early and common feedback from users
- Cumulative costs assessed regularly.

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 evaluation knowledge 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 promise risky because of possible changes to economic priorities.
- Users are unsure of their needs.
- Requirements are complex.
- Significant changes are expected (research and exploration) e.t.c
- When the project is large.
- When reqmts are unclear and complex.

Rational Unified Process

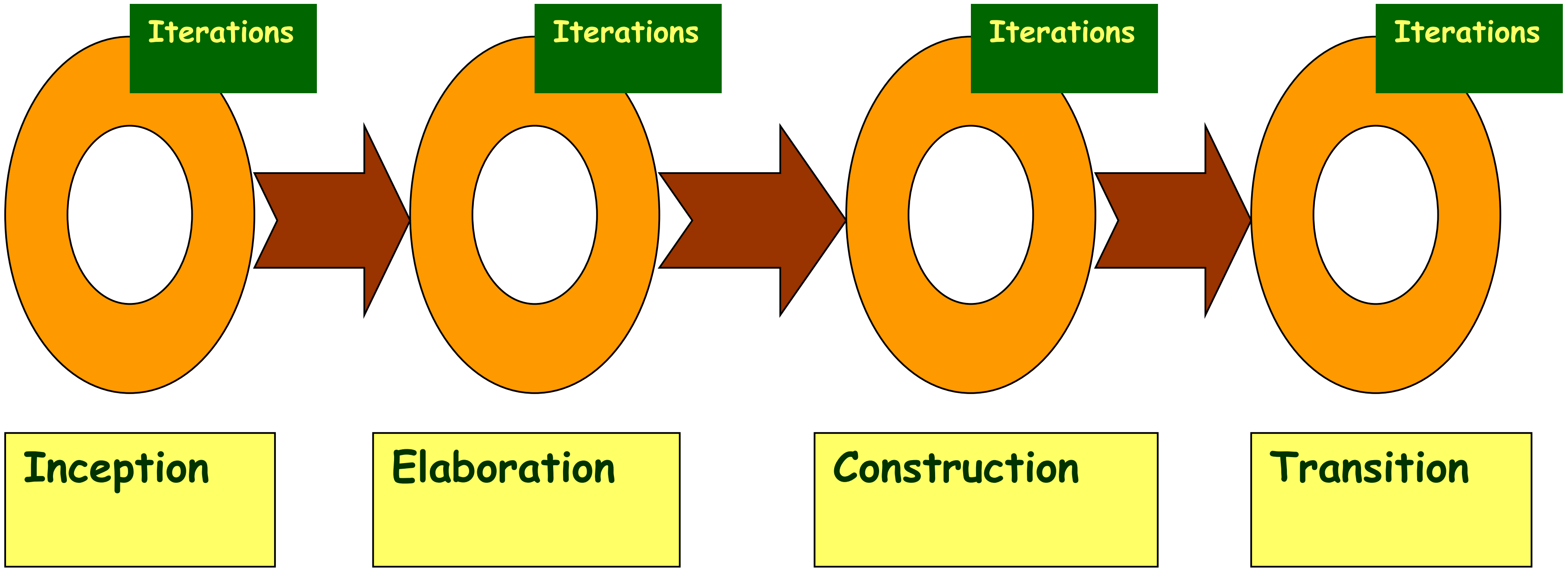
- Originally developed by Rational Software:
  - Obtain by *IBM* in February 2003.
- Based on six best practices:
  - ➢Develop iteratively, with risk as the primary iteration driver.
  - ➢Manage requirements .
  - ➢Use a component-based architecture .
  - ➢Model software visually.
  - ➢Continuously verify quality .
  - ➢Control changes .

Four Phases and iterative Development at Every phase

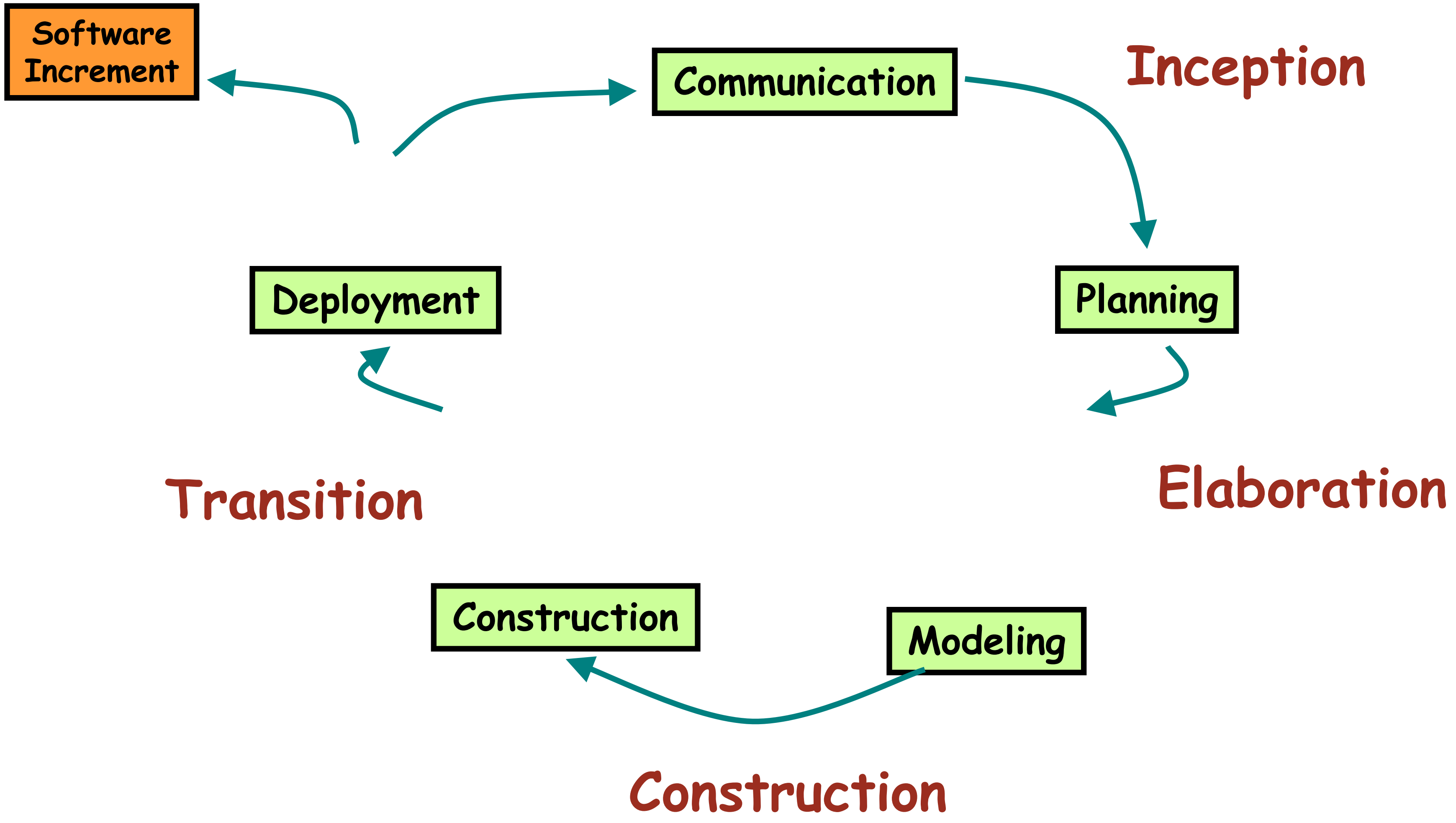
- *Inception Phase*
- *Elaboration Phase*
- *Construction Phase*
- *Transition Phase*

RUP Iterations in Phases

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



Unified process



Unified process work products

Inception phase
Vision Document
Initial Use-case Model
Initial Business Case
Initial Risk List
Project Plan
Prototype(s)
. . .

Elaboration phase
Use-case Model
Requirements
Analysis Model
Preliminary Model
Revised Risk List
Preliminary Manual
...

Construction phase
Design Model
SW Components
Test Plan
Test Procedure
Test Cases
User Manual
Installation Manual
...

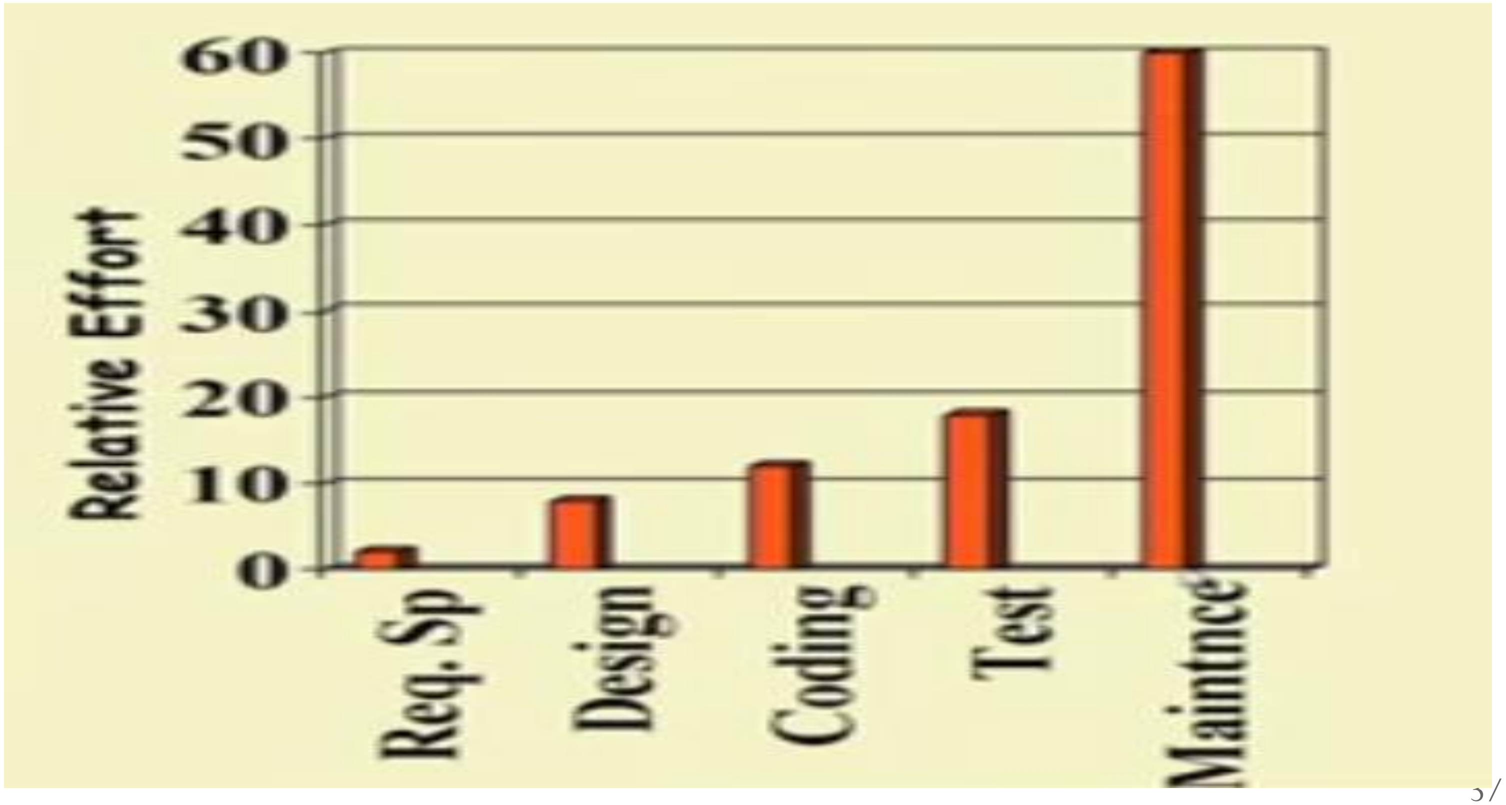
Transition phase
SW Increment
Beta Test Reports
User Feedback
...

Comparison of Different Life Cycle Models

- Iterative waterfall model
  - ✓Most widely used model.
  - ✓But suitable only for well understand problems.
- Prototype model
  - ✓Suitable for projects not well- understood in terms of user reqmts and technical aspects.
- Evolutionary model:
  - ✓Suitable for large problems which can be decompose into a set of modules.
  - ✓Suitable for incremental delivery of the system.
- Spiral model:
  - ✓ Suitable for development of technically challenging software products that are subject to several kinds of risks.

Relative Effort for phases

- Phases between the feasibility study and testing is called development phase.
- Among all life-cycle phases maintenance phase takes maximum effort.
- Among development phase testing takes the maximum effort.



Process Assessment Models

- ✓ A software process assessment is a disciplined examination of the software processes used by an organization, based on a process model.
- ✓ The assessment includes the identification and characterization of current practices, identifying areas of strengths and weaknesses, and the ability of current practices to control or avoid significant causes of poor (software) quality, cost, and schedule. A software assessment (or audit) can be of three types.
- ➢ **Self-assessment (First-party Assessment)** is performed internally by an organization's own personnel.
- ➢ **Second-party Assessment** is performed by an external assessment team or the organization is assessed by a customer.
- ➢ **Third-party Assessment** is performed by an external party or (e.g., a supplier being assessed by a third party to verify its ability to enter contracts with a customer). Software process assessments are performed in an open and collaborative environment.

Software Process Metrics

- ❖ Process metrics are the measures of the development process that creates a body of software.
- ❖ A common example of a process metric is the length of time that the process of software creation task.
- ❖ Process metrics are measurements used to track the performance of a business process.
- ❖ Software metric is a measure of software characteristics which are measurable or countable.
- ❖ Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.
- ❖ Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

Software metrics can be classified into three types as follows:

1. **Product Metrics:** These are the measures of various characteristics of the software product.

- ✓ The two important software characteristics are: **Size** and **complexity** of software and Quality and reliability of software.
- ✓ These metrics can be computed for different stages of SDLC.

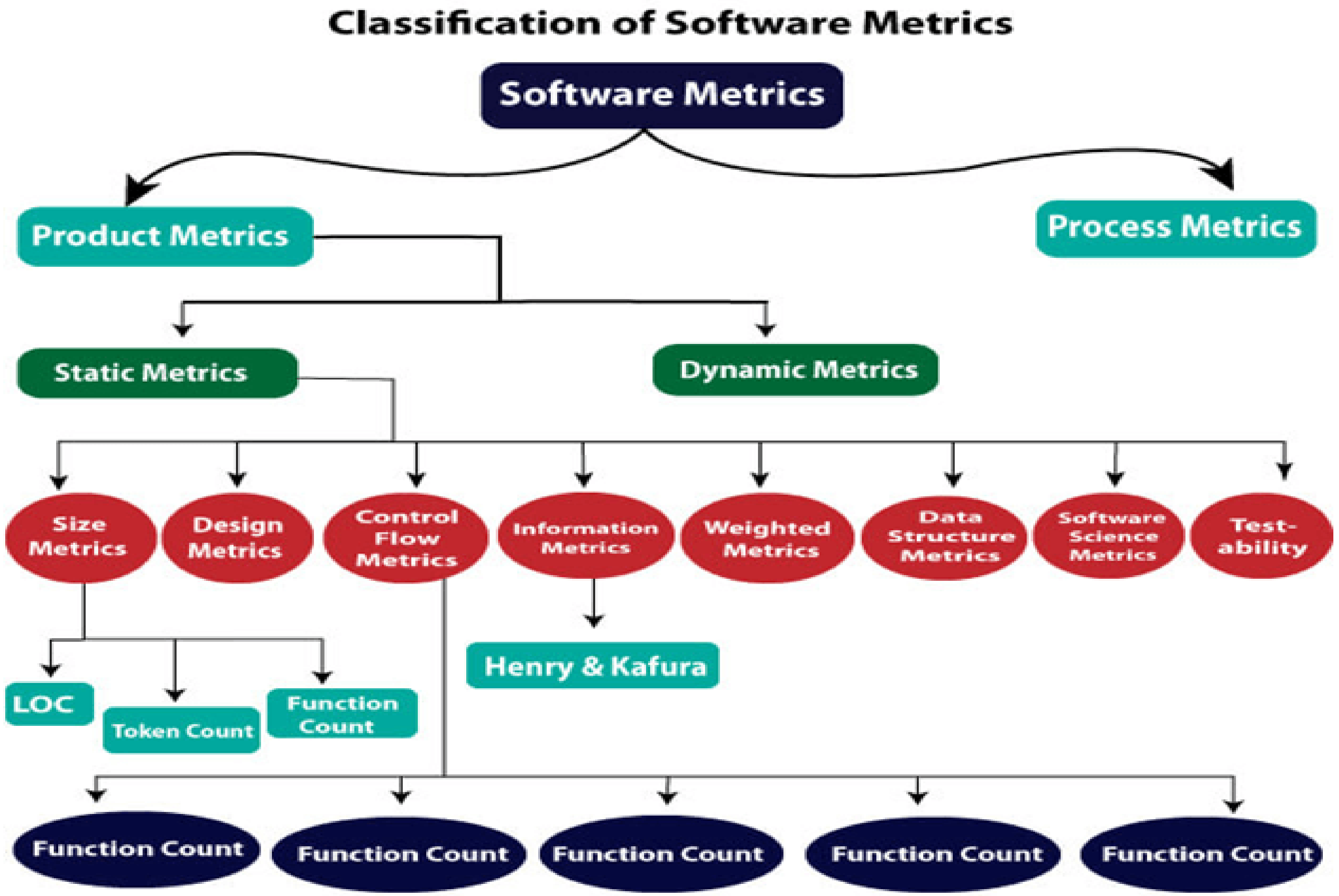
2. **Process Metrics:** These are the measures of various characteristics of the software development process.

- ✓ E.g the efficiency of fault detection.
- ✓ They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.

3. **Project Metrics** describe the project characteristics and execution.

- ✓ Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity

Classification of Software Metrics



An Object Oriented Philosophy

- Object-oriented philosophy is a method of exploring gaps between objects and their components, objects and their appearances, objects and their relations, or objects and their qualities.
- A table is not the same thing as the quarks and electrons of which it is made.
- The OO philosophy suggests that the things manipulated by the program should correspond to things in the real world.
- They should carry the same names as those things carry in the real world.
- They should interact in ways like those objects in the real world.

What have studied now?

For further reading join with this website: OMG website, www.omg.org.