Software Engineering

Module 2: Life Cycle Models

Introduction

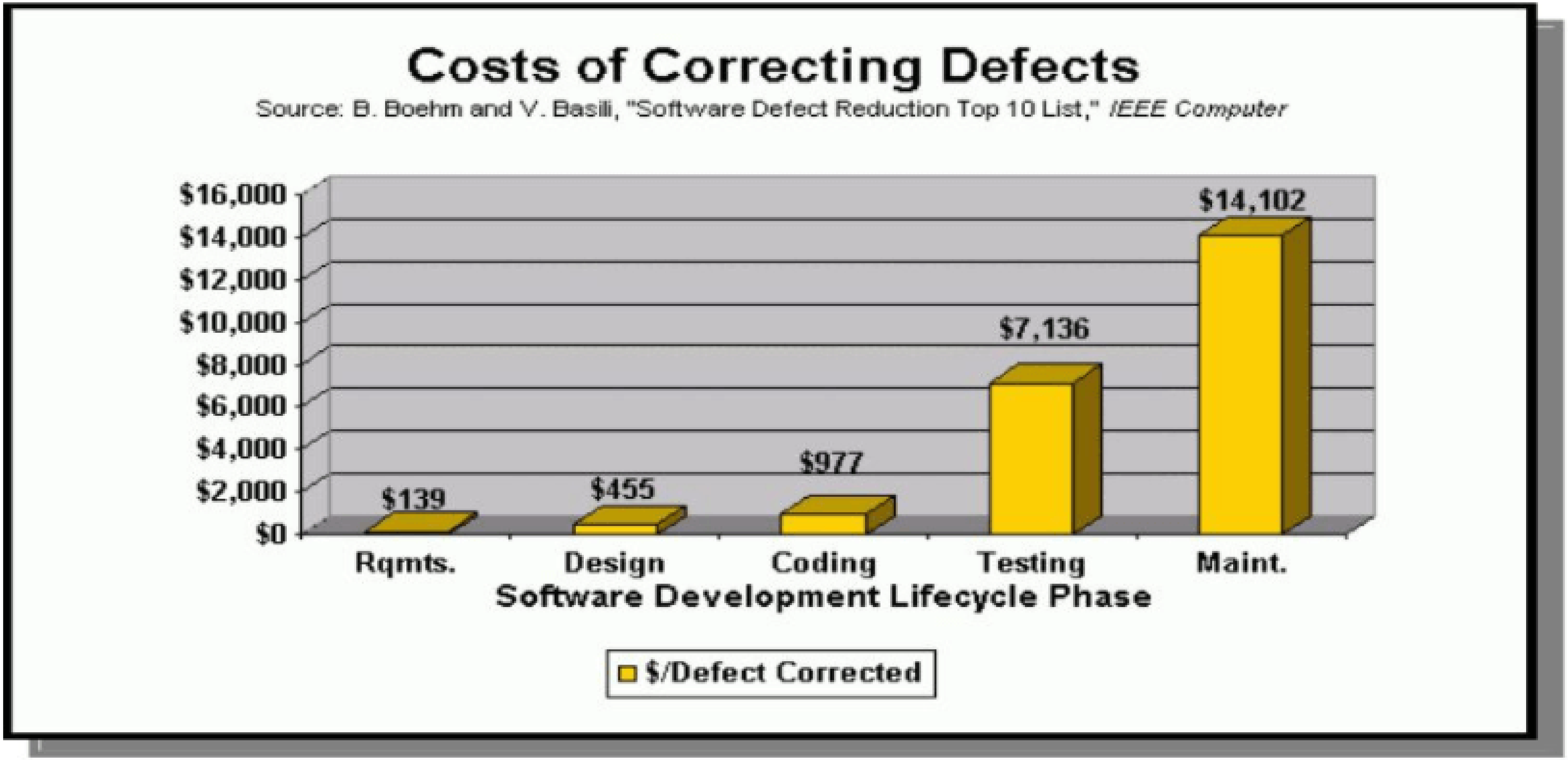
- Emphasis has shifted
  - from error correction to error prevention.
- Modern practices emphasize:
  - detection of errors as close to their point of introduction as possible.
- In exploratory style,
  - errors are detected only during testing,
- Now,
  - focus is on detecting as many errors as possible in each phase of development.

Introduction

- A typical distribution of error occurrences by is:

- Requirement Analysis - 20%
- Design - 30%
- Coding - 50%
- The cost of correcting different phases is not the same and depends on when the error is detected and corrected.
- As one old expect, the greater the delay in detecting an error after it occurs, the more expensive it is to correct it.

Introduction



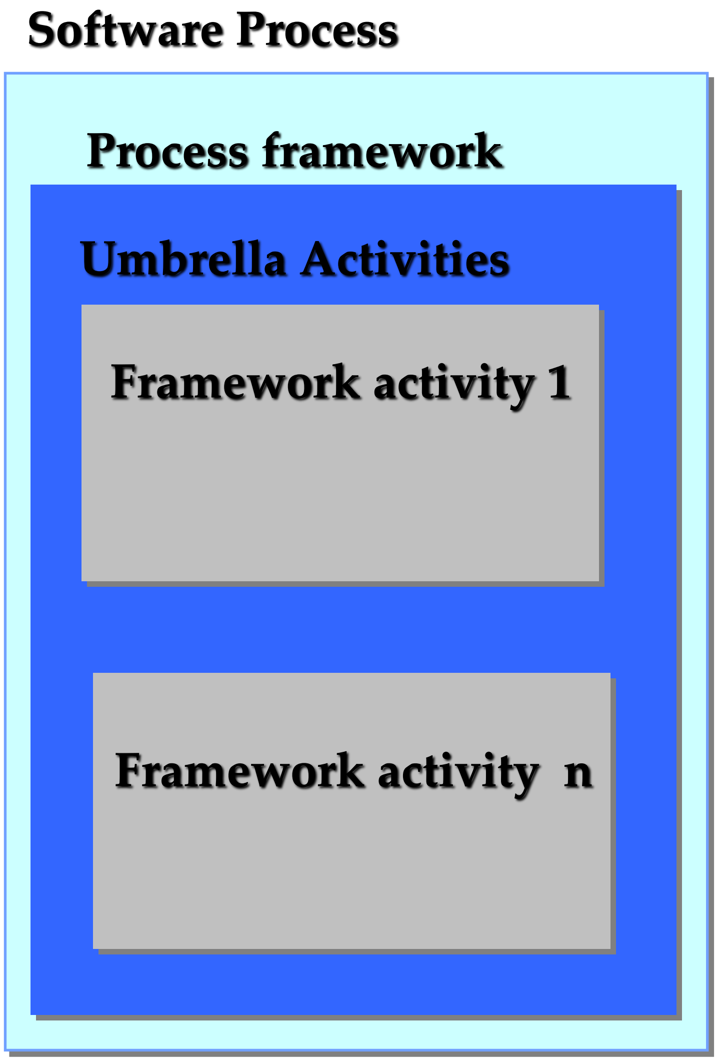
Introduction

- A lot of effort and attention is now being paid to:
  - Requirements specification.
- Also, now there is a distinct design phase:
- Standard design techniques are being used.
- During all stages of development process:
  - Periodic reviews are being carried out
- Software testing has become systematic:
  - Standard testing techniques are available.

Software Process

- **What? A software process** – as a framework for the tasks that are required to build high-quality software.
- **Who?** Managers, software engineers, and customers.
- **Why?** Provides stability, control, and organization to an otherwise chaotic activity.
- **Steps?** A handful of activities are common to all software processes, details vary.
- **Work product?** Programs, documents, and data.

Process Framework



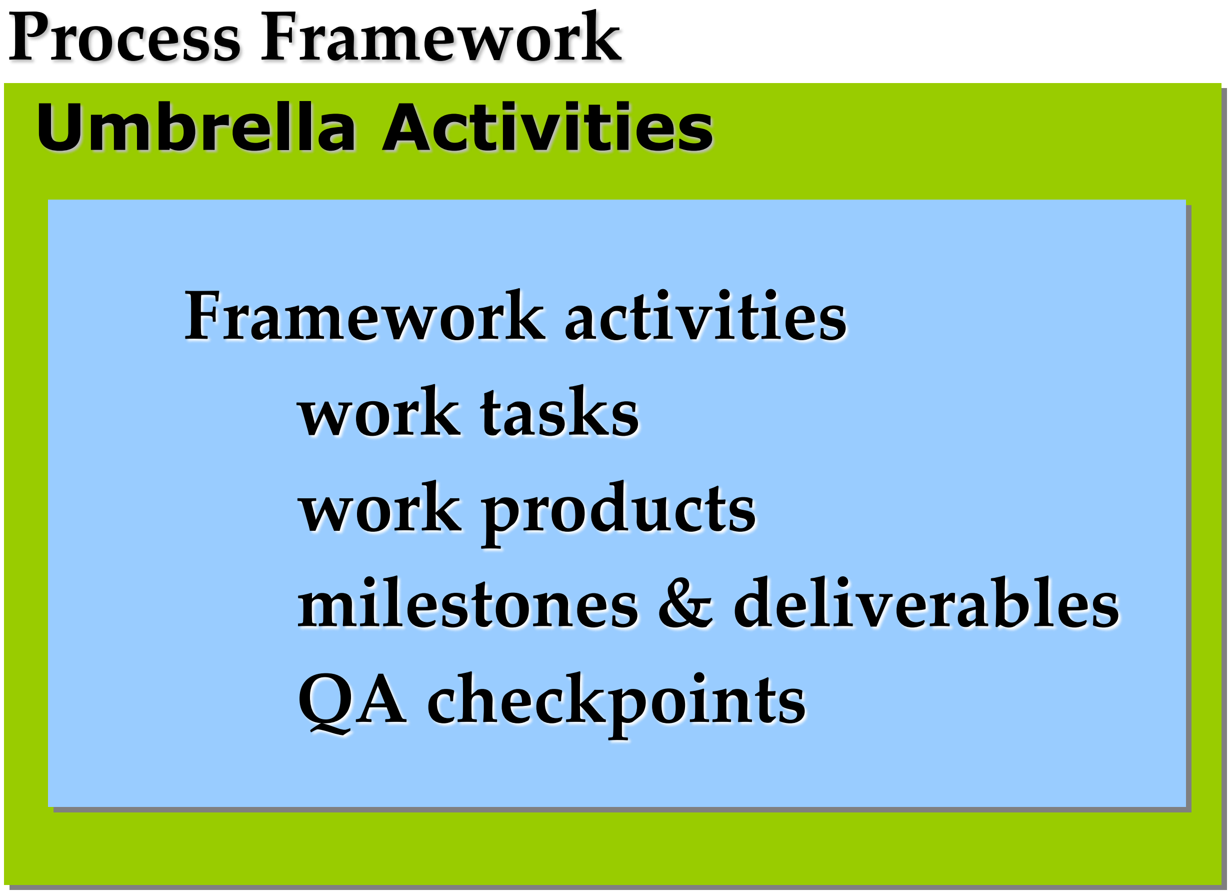
Software Process

Process framework

Umbrella Activities

Framework activity 1

Framework activity n



Process Framework

Umbrella Activities

Framework activities

work tasks

work products

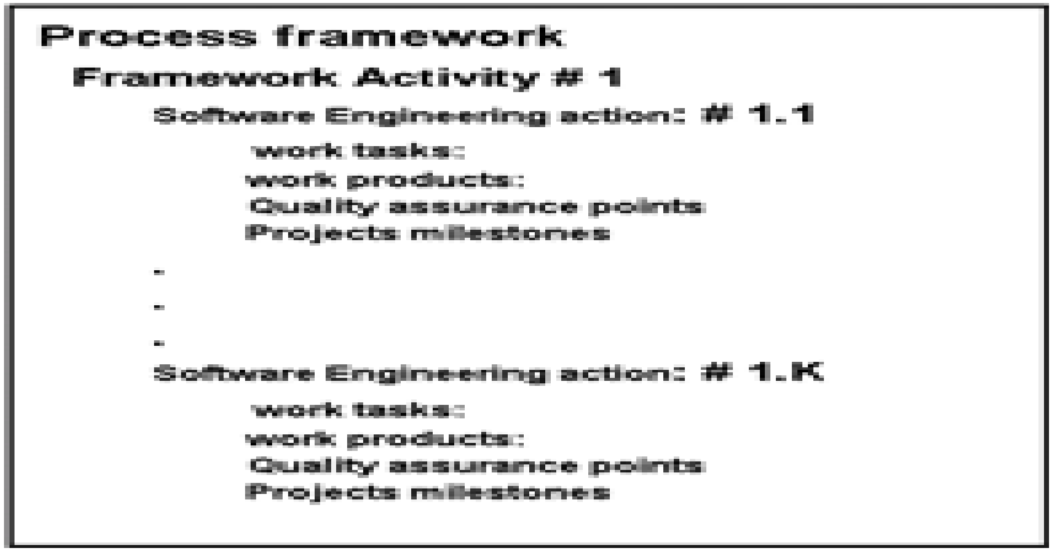
milestones & deliverables

QA checkpoints

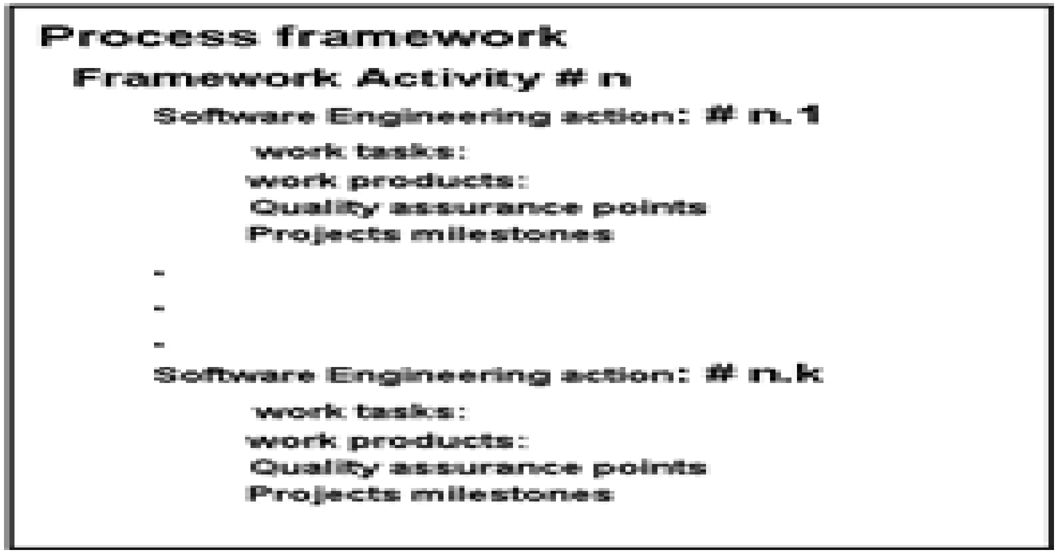
Process framework

- Why process :
- A process defines who is doing what, when and how to reach a certain goal.
- To build complete software process.
- Identified a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.
- It encompasses a set of umbrella activities that are applicable across the entire software process.

Process Framework



Process framework
Framework Activity # 1
Software Engineering action: # 1.1
work tasks:
work products:
Quality assurance points
Projects milestones
-
-
-
Software Engineering action: # 1.K
work tasks:
work products:
Quality assurance points
Projects milestones



Process framework
Framework Activity # n
Software Engineering action: # n.1
work tasks:
work products:
Quality assurance points
Projects milestones
-
-
-
Software Engineering action: # n.k
work tasks:
work products:
Quality assurance points
Projects milestones

•Each framework activities is populated by a set for *software engineering actions* – a collection of related tasks.

• Each action has individual *work task*.

Generic Process Framework Activities

- Communication:
  - Heavy communication with customers, stakeholders, team
  - Encompasses requirements gathering and related activities
- Planning:
  - Workflow that is to follow
  - Describe technical task, likely risk, resources will require, work products to be produced and a work schedule.
- Modeling:
  - Help developer and customer to understand requirements (Analysis of requirements) & Design of software
- Construction
  - Code generation: either manual or automated or both
  - Testing – to uncover error in the code.
- Deployment:
  - Delivery to the customer for evaluation
  - Customer provide feedback

The Process Model: Adaptability

- The framework activities will always be applied on every project ... BUT
- The tasks for each activity will vary based on:
  - The type of project (an “entry point” to the model)
  - Characteristics of the project
  - Common sense judgment; concurrence of the project team

Umbrella Activities

- Software project tracking and control
  - Assessing progress against the project plan.
  - Take adequate action to maintain schedule.
- Formal technical reviews
  - Assessing software work products in an effort to uncover and remove errors before goes into next action or activity.
- Software quality assurance
  - Define and conducts the activities required to ensure software quality.
- Software configuration management
  - Manages the effects of change.
- Document preparation and production
  - Help to create work products such as models, documents, logs, form and list.
- Reusability management
  - Define criteria for work product reuse
  - Mechanisms to achieve reusable components.
- Measurement
  - Define and collects process, project, and product measures
  - Assist the team in delivering software that meets customer's needs.
- Risk management
  - Assesses risks that may effect that outcome of project or quality of product (i.e. software)

Life Cycle Model

- A software life cycle model (or process model):
  - a descriptive and diagrammatic model of software life cycle
  - identifies all the activities required for product development,
  - establishes a precedence ordering among the different activities,
  - Divides life cycle into phases.

Software Life Cycle

- 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.

Why Model Life Cycle ?

- A written description:
  - Forms a common understanding of activities among the software developers.
  - Helps in identifying inconsistencies, redundancies, and omissions in the development process.
  - Helps in tailoring a process model for specific projects.
- 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 ---
  - he has the freedom to decide his exact steps.
- When a software product 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.

Life Cycle Model (CONT.)

- 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.
- 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.

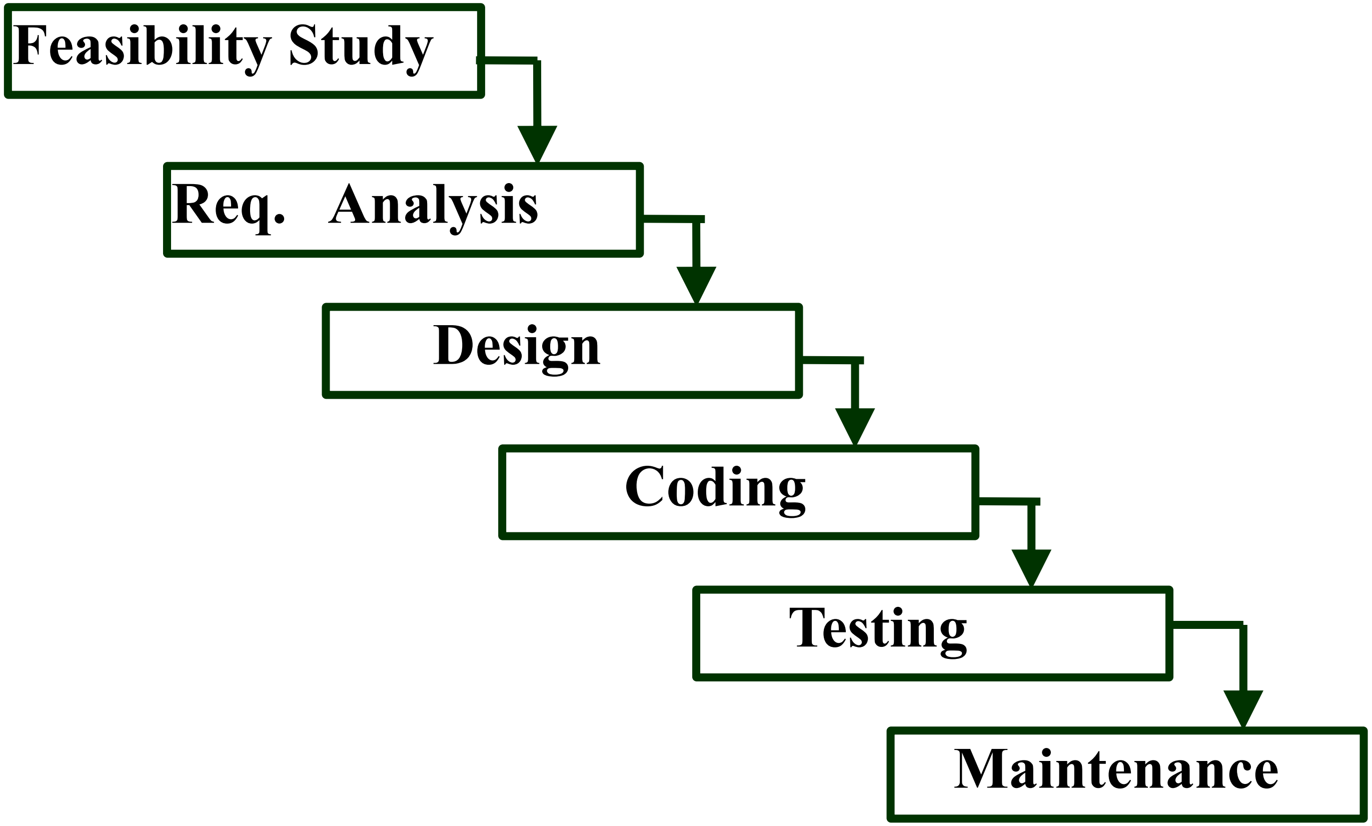
Life Cycle Model (CONT.)

- Many life cycle models have been proposed.
- We will confine our attention to a few important and commonly used models.
  - Classical waterfall model,
  - Iterative waterfall,
  - Evolutionary,
  - Prototyping, and
  - Spiral model
  - Agile (XP, Scrum, TDD, etc.)

Classical Waterfall Model

- Classical waterfall model divides life cycle into phases:
  - feasibility study,
  - requirements analysis and specification,
  - design,
  - coding and unit testing,
  - integration and system testing,
  - maintenance.

Classical Waterfall Model


Feasibility Study

Req. Analysis

Design

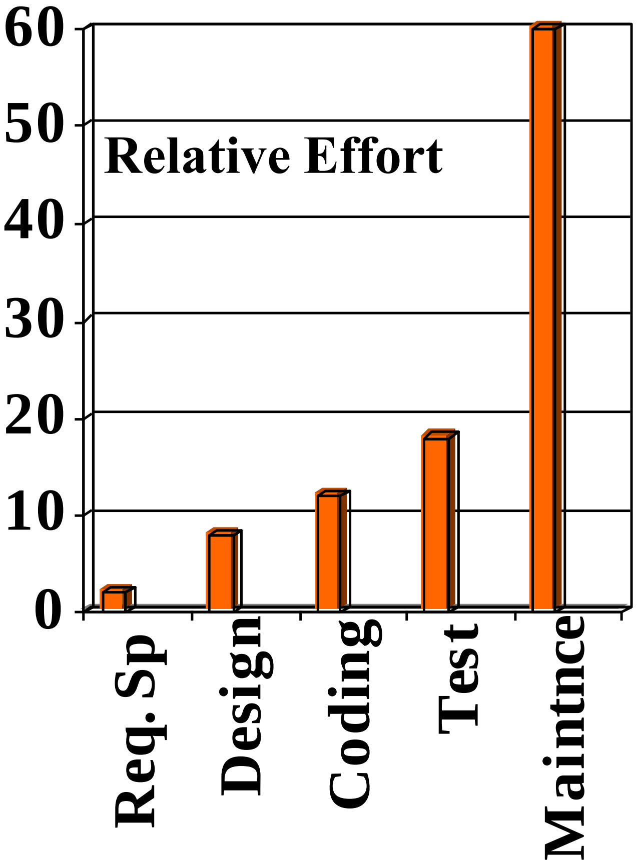
Coding

Testing

Maintenance

Relative Effort for Phases

- Phases between feasibility study and testing
  - known as development phases.
- Among all life cycle phases
  - maintenance phase consumes maximum effort.
- Among development phases,
  - testing phase consumes the maximum effort.



Classical Waterfall Model
(CONT.)

- 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 specific methodologies for:
  - specification,
  - design,
  - testing,
  - project management, etc.

Feasibility Study

- Main aim of feasibility study:determine whether developing the product
  - financially worthwhile
  - technically feasible.
- First roughly understand what the customer wants:
  - different 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.

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.

Activities during Feasibility Study

- Perform a cost/benefit analysis:
  - to determine which solution is the best.
  - you may determine that none of the solutions is feasible due to:
    - high cost,
    - resource constraints,
    - technical reasons.

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.

Goals of Requirements Analysis

- Collect all related data from the customer:
  - analyze the collected data to clearly understand what the customer wants,
  - find out any inconsistencies and incompleteness in the requirements,
  - resolve all inconsistencies and incompleteness.

Requirements Gathering

- Gathering relevant data:
  - usually collected from the end-users through interviews and discussions.
  - For 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:
  - would usually contain several contradictions and ambiguities:
  - each user typically has only a partial and incomplete view of the system.
- Ambiguities and contradictions:
  - must be identified
  - resolved by discussions with the customers.
- Next, requirements are organized:
  - into a Software Requirements Specification (SRS) document.

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.
- In technical terms:
  - during design phase, software architecture is derived from the SRS document.
- Two design approaches:
  - traditional approach,
  - object oriented approach.

Traditional Design Approach

- Identify all the functions to be performed.
- Identify data flow among the functions.
- Decompose each function recursively into sub-functions.
  - Identify data flow among the sub functions as well.
- Carried out using Data flow diagrams (DFDs).
- After structured analysis, carry out structured design:
  - architectural design (or high-level design)
  - detailed design (or low-level design).

Object Oriented Design

- 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 structure
  - further refined to obtain the detailed design.

Implementation

- Purpose of implementation phase
  - translate software design into source code.
- During the implementation phase:
  - each module of the design is coded,
  - each module is unit tested
    - tested independently as a stand alone unit, and debugged,
  - each module is documented.

Implementation (CONT.)

- The purpose of unit testing:
  - test if individual modules work correctly.
- The end product of implementation phase:
  - a set of program modules that have been tested individually.

Integration and System Testing

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out 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.

Maintenance

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

Maintenance (CONT.)

- 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

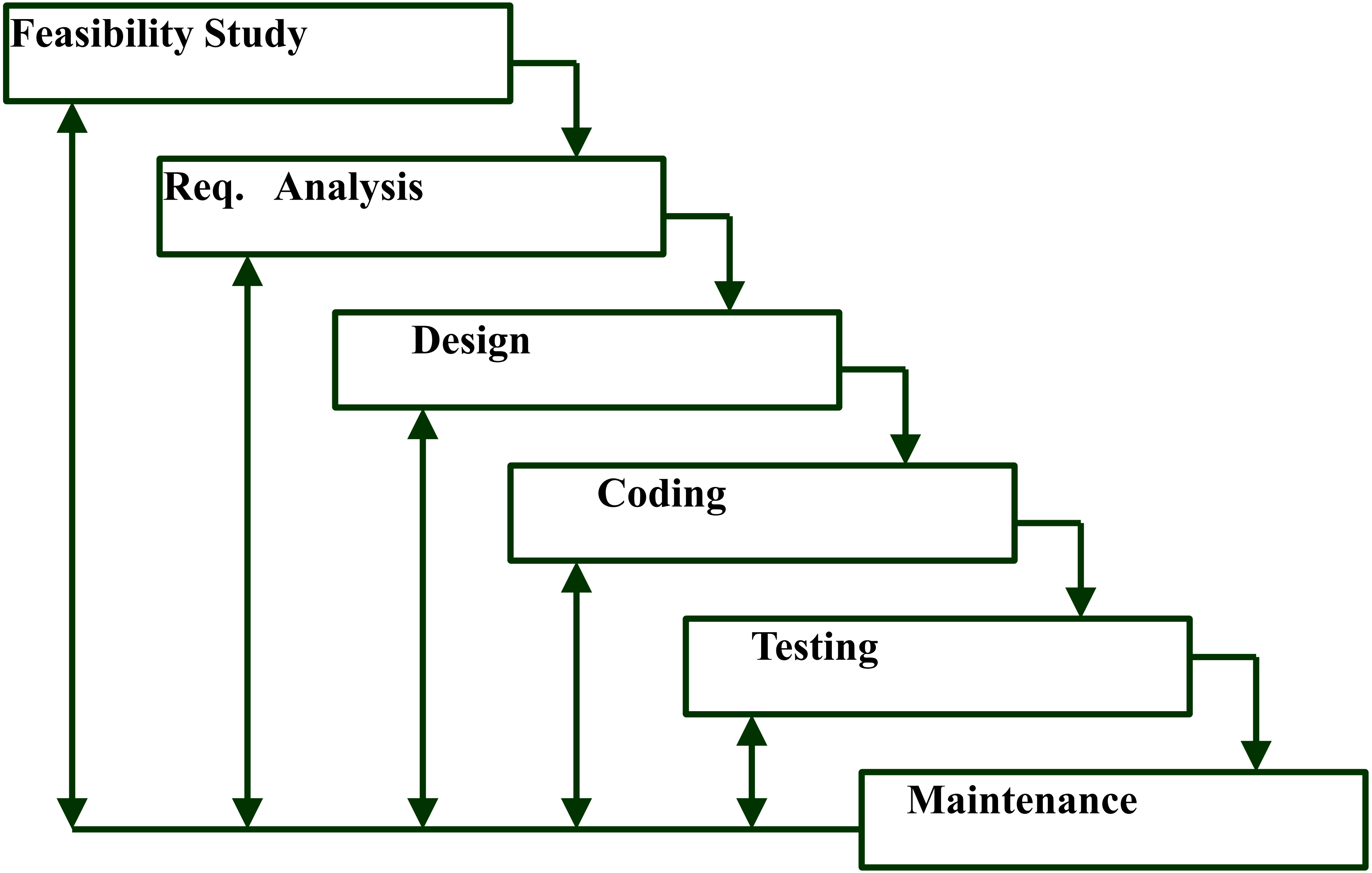
- Classical waterfall model is idealistic:
  - assumes that no defect is introduced during any development activity.
  - in practice:
    - defects do get introduced in almost every phase of the life cycle.
- Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase.

Iterative Waterfall Model

(CONT.)

- Once a defect is detected:
  - we need to go back to the phase where it was introduced
  - redo some of the work done during that and all subsequent phases.
- Therefore we need feedback paths in the classical waterfall model.

Iterative Waterfall Model



Feasibility Study

Req. Analysis

Design

Coding

Testing

Maintenance

Iterative Waterfall Model

- 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.**

Iterative Waterfall Model

- Iterative waterfall model is by far the most widely used model.
  - Almost every other model is derived from the waterfall model.
- Irrespective of the life cycle model actually followed:
  - the documents should reflect a classical waterfall model of development,
  - comprehension of the documents is facilitated.

Prototyping 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 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.

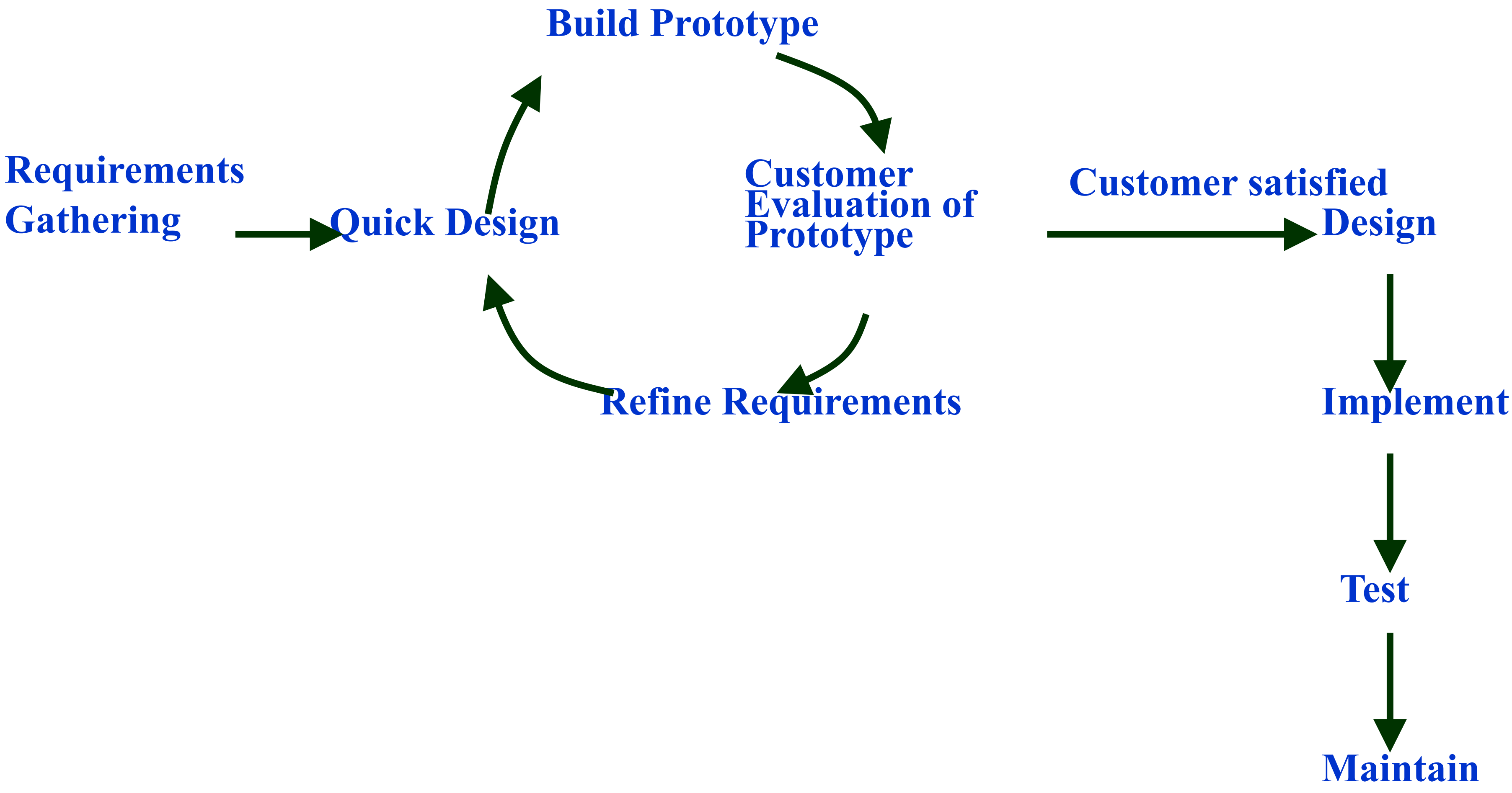
Prototyping Model (CONT.)

- The third reason for developing a prototype is:
  - it is impossible to “get it right” the first time,
  - we must plan to throw away the first product
    - if we want to develop a good product.

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 function may use 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, requirements are refined.
  - This cycle continues until the user approves the prototype.
- The actual system is developed using the classical waterfall approach.

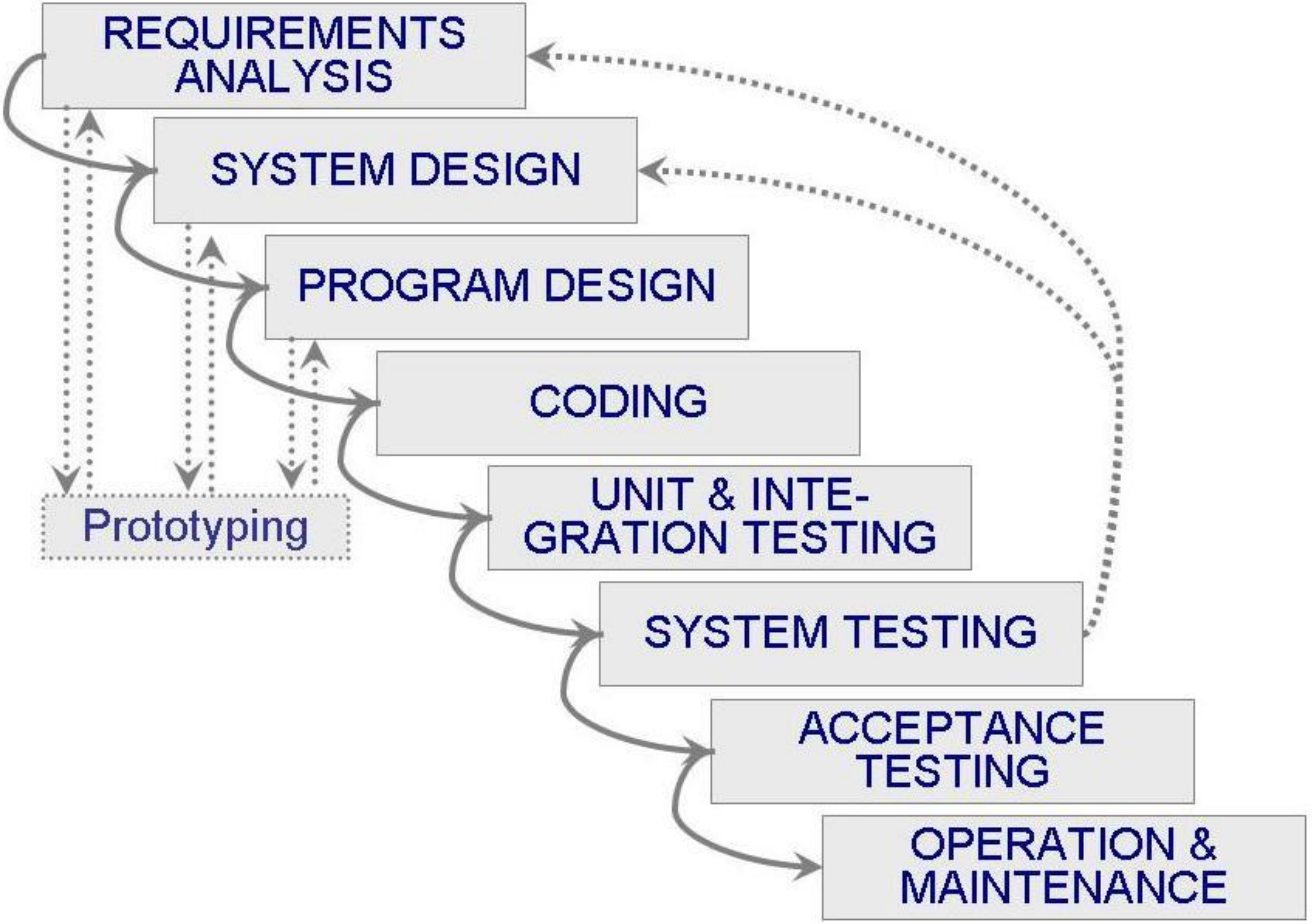
Prototyping Model (CONT.)

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

Prototyping Model (CONT.)

- Even though construction of a working prototype model involves additional cost --- overall development cost might be 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.**

Waterfall Model with Prototype



REQUIREMENTS ANALYSIS

SYSTEM DESIGN

PROGRAM DESIGN

CODING

UNIT & INTEGRATION TESTING

SYSTEM TESTING

ACCEPTANCE TESTING

OPERATION & MAINTENANCE

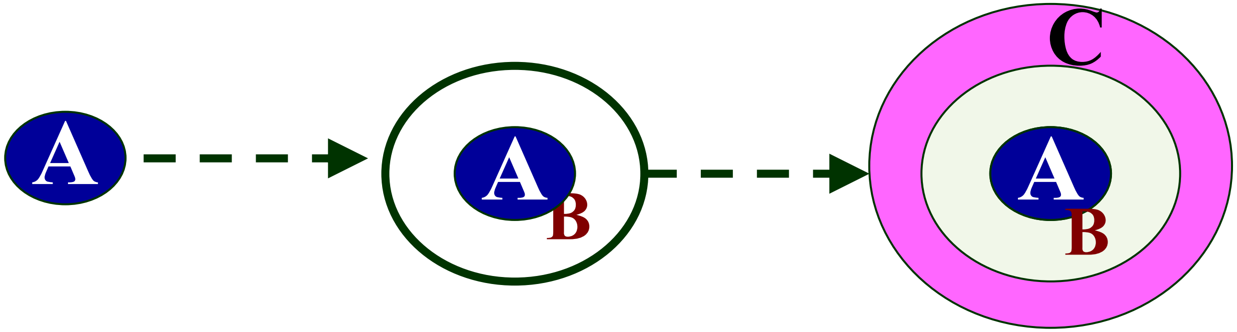
Prototyping

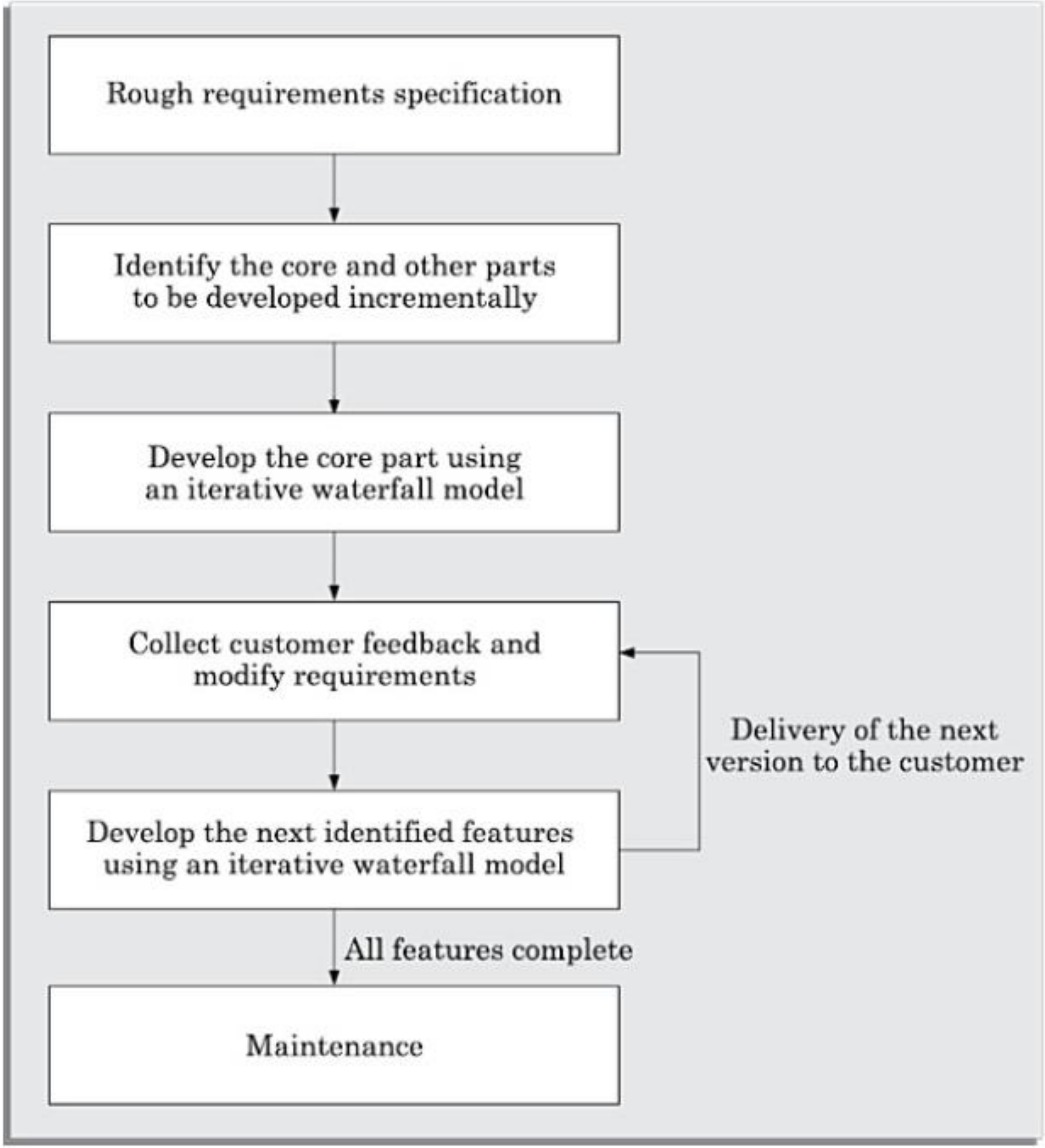
Evolutionary Model

- Evolutionary model (successive versions or incremental model):
  - The system is broken down into several modules which can be incrementally implemented and delivered.
  - The requirements, plan, estimates, and solution evolve over the iterations, rather than fully defined and frozen specification effort before.
  - the development iterations begin.
- First develop the core modules of the system.
- The initial product skeleton is refined into increasing levels of capability:
  - by adding new functionalities in successive versions.

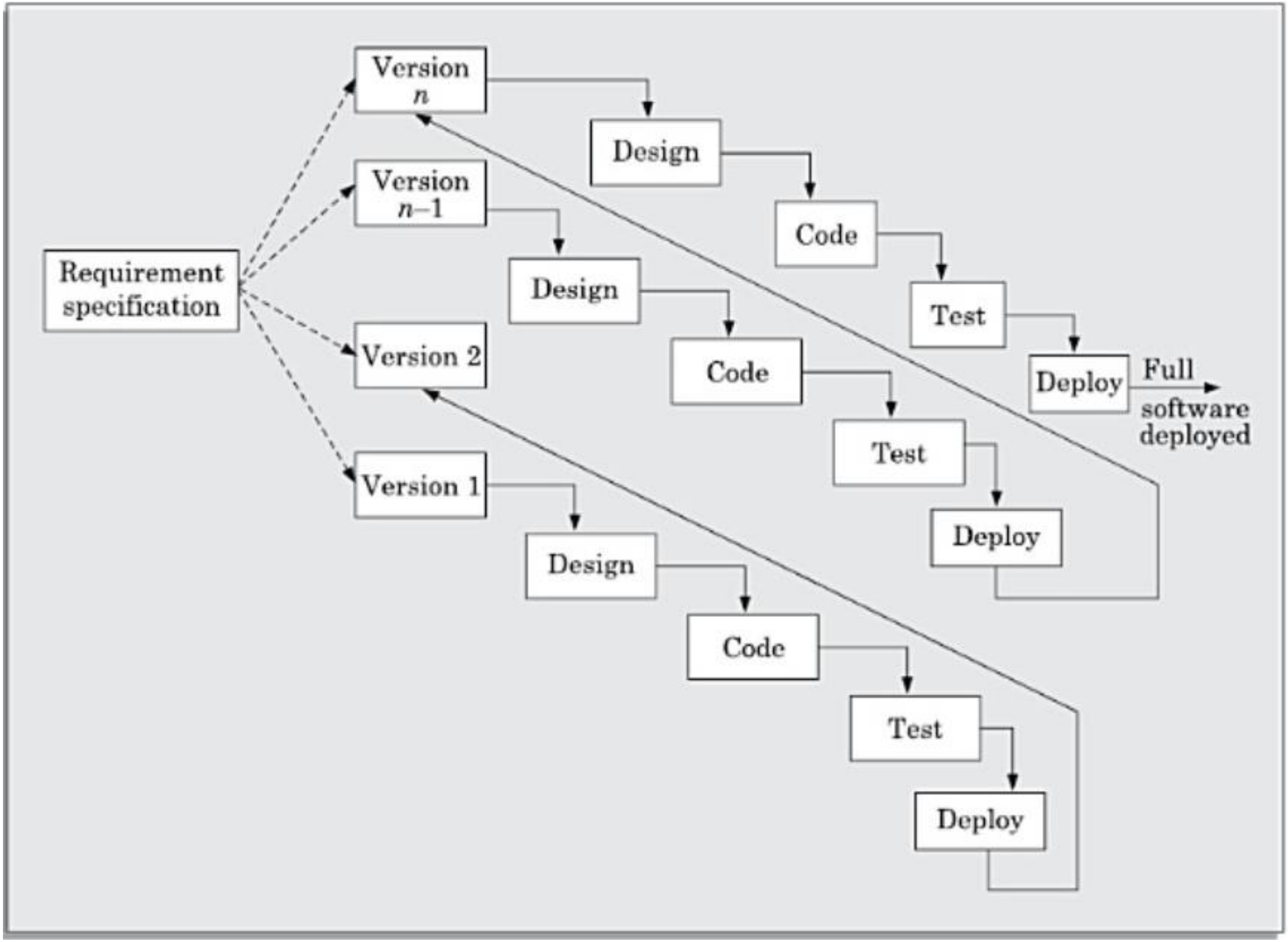
Evolutionary Model (CONT.)

- Successive version of the product:
  - 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.




Rough requirements specification
Identify the core and other parts to be developed incrementally
Develop the core part using an iterative waterfall model
Collect customer feedback and modify requirements
Delivery of the next version to the customer
Develop the next identified features using an iterative waterfall model
All features complete
Maintenance

Evolutionary Model Vs Incremental



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 thoroughly:
  - reduces chances of errors in final product.
- Easy handling change requests
- Incremental resource deployment

Disadvantages of Evolutionary Model

- Often, difficult to subdivide problems into functional units:
  - which can be incrementally implemented and delivered.
  - evolutionary model is useful for very large problems,
    - where it is easier to find modules for incremental implementation.

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 exact number of phases through which the product is developed can be varied by the project manager depending upon the project risks.

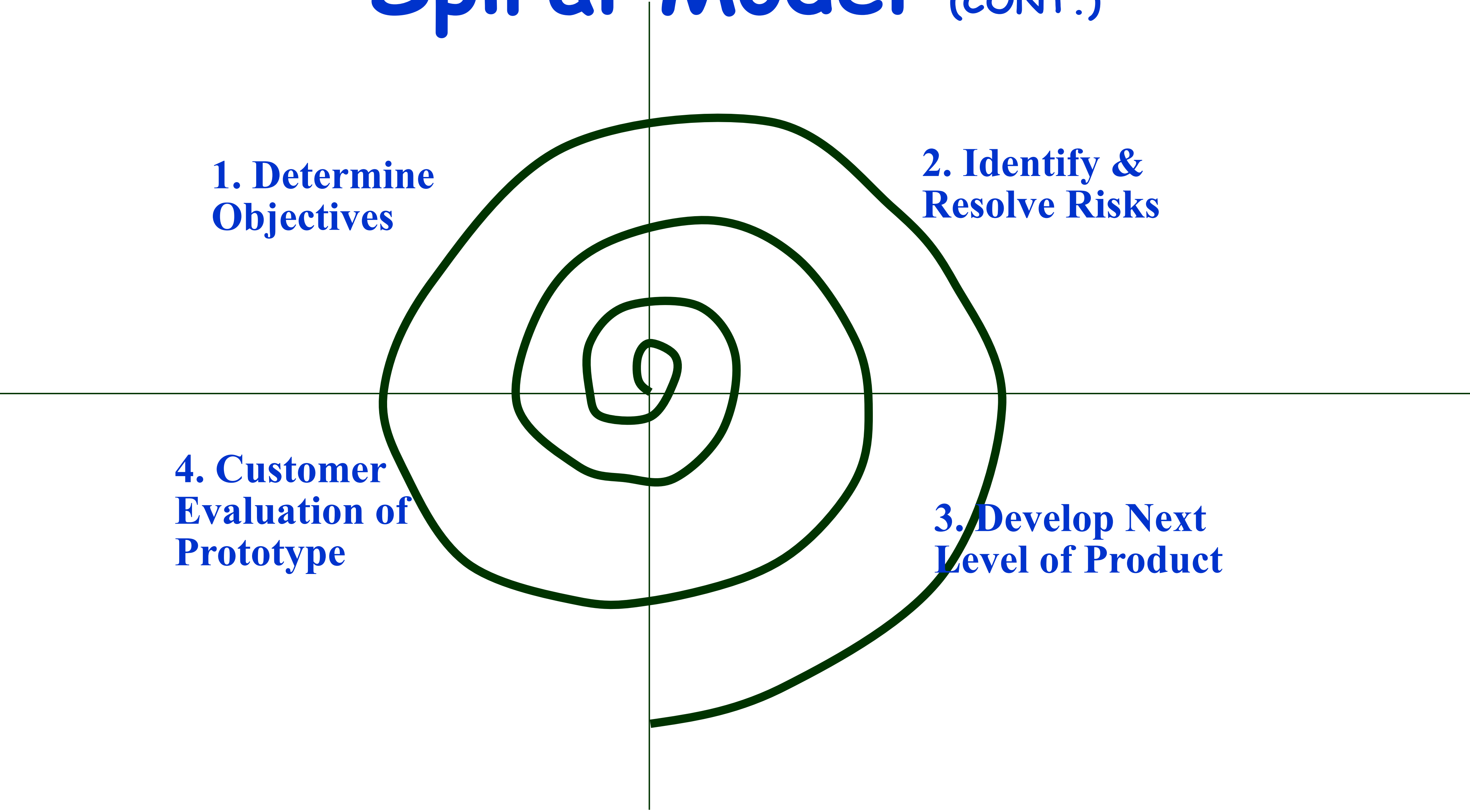
Spiral Model

- A prominent feature of the spiral model is handling unforeseen risks that can show up much after the project has started.
- Prototyping model can be used effectively only when the risks in a project can be identified upfront before the development work starts.
- It assumed that all risks have been identified completely before the project start.
- In the spiral model prototypes are built at the start of every phase.

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.)



1. Determine Objectives

2. Identify & Resolve Risks

3. Develop Next Level of Product

4. Customer Evaluation of Prototype

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 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 Model as a meta model

- Subsumes all discussed models:
  - a single loop spiral represents waterfall model.
  - uses an evolutionary approach --
    - iterations through 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.

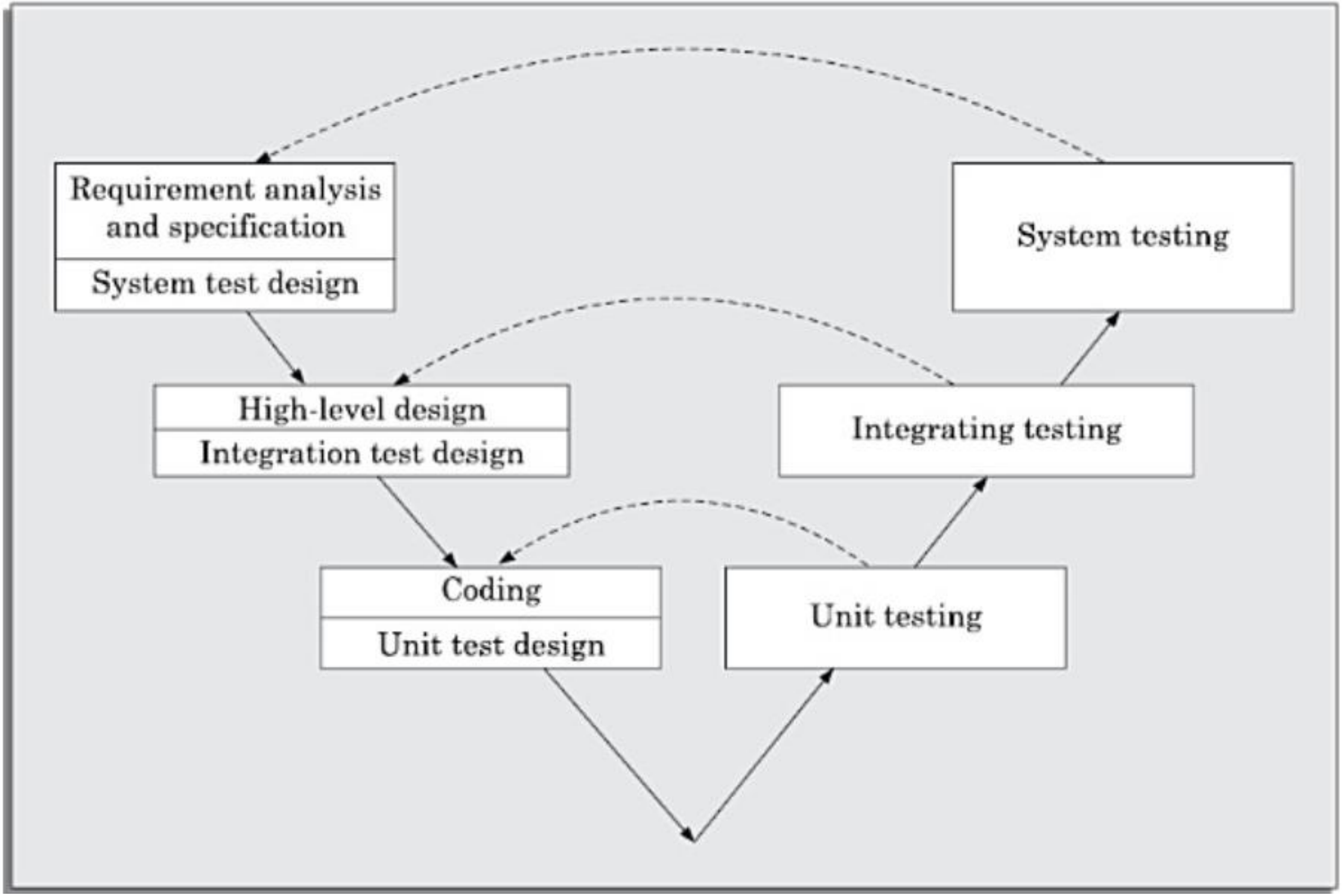
Spiral Model as a meta model

- For projects having many unknown risks that might show up as the development proceeds, the spiral model would be the most appropriate development model to follow.
- Spiral model that restrict its use to a only a few types of projects.
- The spiral model usually appears as a complex model to follow, since it is risk driven and is more complicated phase structure than the other models.

V-Model

- V-model is a variant of the waterfall model.
- *As* is the case with the waterfall model, this model gets its name from its visual appearance.
- In this model verification and validation activities are carried out throughout the development life cycle, and therefore the chances bugs in the work products considerably reduce.
- This model is therefore generally considered to be suitable for use in projects concerned with development of *safety-critical software* that are required to have *high reliability.*

V-Model


Requirement analysis and specification
System test design
System testing
High-level design
Integration test design
Integrating testing
Coding
Unit test design
Unit testing

V-Model

- In each development phase, along with the development of a work product, test case design and the plan for testing the work product are carried out, whereas the actual testing is carried out in the validation phase.

- This validation plan created during the development phases is carried out in the corresponding validation phase.

- In the validation phase, testing is carried out in three steps—unit, integration, and system testing.

Advantages of V-Model

- Much of the testing activities (test case design, test planning, etc.) are carried out in parallel with the development activities.
- Usually leads to a shorter testing phase and an overall faster product development as compared to the iterative model.
- The test team is associated with the project from the beginning.
- Therefore they build up a good understanding of the development artifacts, and this in turn, helps them to carry out effective testing of the software.

Disadvantages of V-Model

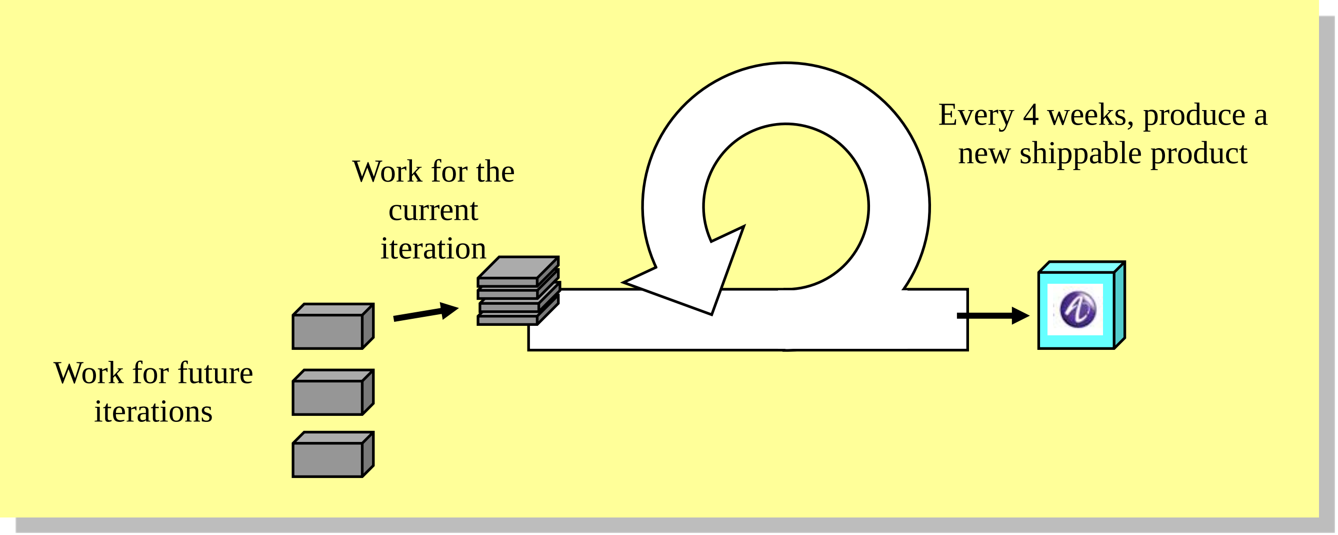
- Being a derivative of the classical waterfall model, this model inherits most of the weaknesses of the waterfall model.

Agile Model

- Agile software development is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
  - Methods
  - Iterative
  - incremental
- It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change.

Agile Model

- Agile Development as a “software development framework” says:
  - keep things small
  - deliver partially-completed software frequently
  - talk to the customer often
  - write more code than documentation
  - everyone on the team learns together


Every 4 weeks, produce a new shippable product
Work for the current iteration
Work for future iterations

Agile Model

- There are many Agile practices:
  - short time-boxed iterations
  - continuous integration
  - daily unit testing
  - regular retrospectives
  - direct communication between developers and the customer or a customer surrogate
  - a single list of features and tasks
  - short-term estimation of development tasks
  - information radiators
  - refactoring
- Will you use every Agile practice? Maybe not.... they are not all required.

Agile Model

- Agile model emphasize face-to-face communication over written documents.
- It is recommended that the development team size be deliberately kept small (5–9 people).
- This helps the team members meaningfully engage in face-to-face communication and have collaborative work environment.
- It is implicit then that the agile model is suited to the development of small projects.
- Its working principle is "*design a little, build a little, test a little, deploy a little*".

Agile Model

- Agile development projects usually deploy pair programming.
- In pair programming, two programmers work together at one work station.
- One types in code while the other reviews the code as it is typed in.
- The two programmers switch their roles every hour or so.

Disadvantages of Agile Model

- Lack of formal documents leaves scope for confusion and important decisions taken during different phases can be misinterpreted at later points of time by different team members.
- In the absence of any formal documents, it becomes difficult to get important project decisions such as design decisions to be reviewed by external experts.
- When the project completes and the developers disperse, maintenance can become a problem.

Extreme Programming (XP)

- Extreme programming (XP) is an important process model under the agile umbrella and was proposed by Kent Beck in 1999.

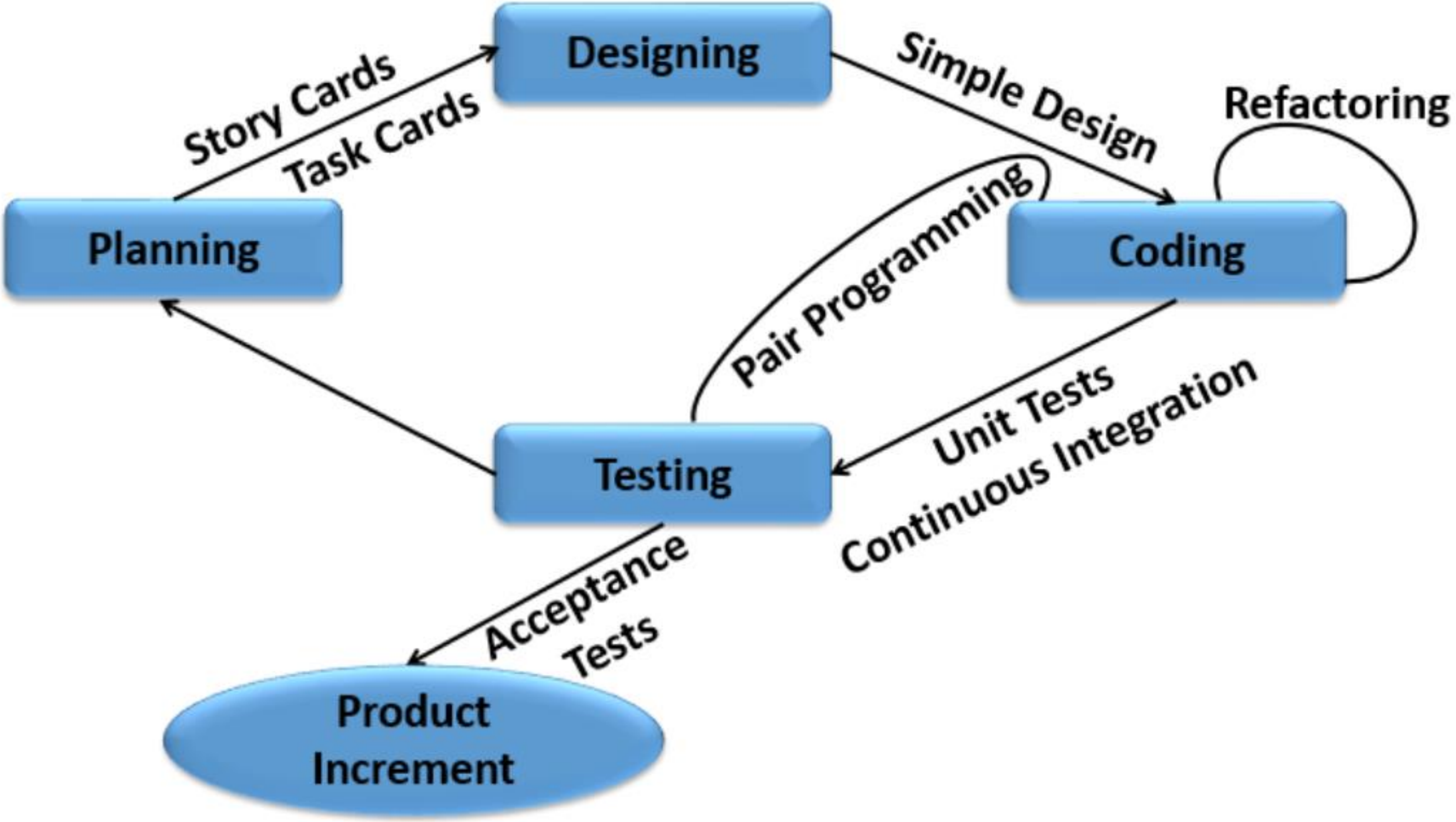
- XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

- eXtreme Programming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming (XP)

- XP is based on frequent releases, during which the developers implement “user stories”.
- A user story is the conversational description by the user about a feature of the required system.
- On the basis of user stories, the project team proposes “metaphors”—a common vision of how the system would work.
- The development team may decide to construct a *spike* for some feature.
- A *spike*, is a very simple program that is constructed to explore the suitability of a solution being proposed.

Practices in XP

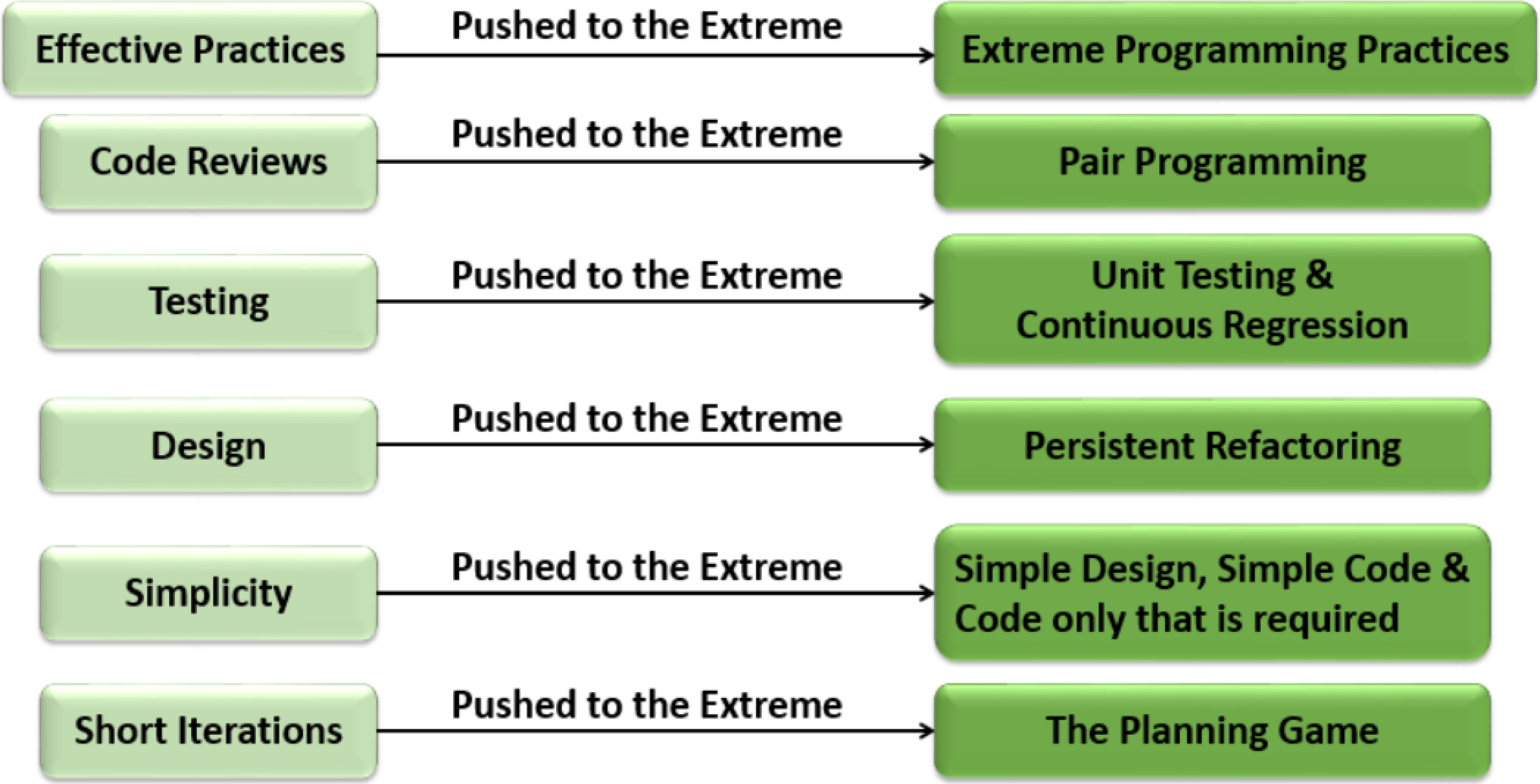


Practices in XP

- **Code review:** The programmers take turn in writing programs and while one writes the other reviews code that is being written.
- **Testing:** XP suggests *test-driven development* (TDD) to continually write and execute test cases.
- **Incremental development:** It suggests that the team should come up with new increments every few days.
- **Simplicity:** For creating the simplest code, one can ignore the aspects such as efficiency, reliability, maintainability, etc.
- **Design:** This can be achieved through *refactoring*, whereby a working code is improved for efficiency and maintainability.
- **Integration testing:** XP suggests that the developers should achieve continuous integration, by building and performing integration testing several times a day.

Why is it called "Extreme?"

- Extreme Programming takes the effective principles and practices to extreme levels.



Effective Practices	Pushed to the Extreme	Extreme Programming Practices
Code Reviews	Pushed to the Extreme	Pair Programming
Testing	Pushed to the Extreme	Unit Testing & Continuous Regression
Design	Pushed to the Extreme	Persistent Refactoring
Simplicity	Pushed to the Extreme	Simple Design, Simple Code & Code only that is required
Short Iterations	Pushed to the Extreme	The Planning Game

Extreme Programming Advantages

- **Slipped schedules:** Short and achievable development cycles ensure timely deliveries.
- **Cancelled projects:** Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.
- **Costs incurred in changes:** Extensive and ongoing testing makes sure the changes do not break the existing functionality.
  - A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected.
- **Production and post-delivery defects:** Emphasis is on the unit tests to detect and fix the defects early.

Extreme Programming Advantages

- **Misunderstanding the business and/or domain:** Making the customer a part of the team ensures constant communication and clarifications.
- **Business changes:** Changes are considered to be inevitable and are accommodated at any point of time.
- **Staff turnover:** Intensive team collaboration ensures enthusiasm and good will.
  - Cohesion of multi-disciplines fosters the team spirit.

Applicability of XP

- **Projects involving new technology or research projects:** In this case, the requirements change rapidly and unforeseen technical problems need to be resolved.
- **Small projects:** Extreme programming was proposed in the context of small teams as face to face meeting is easier to achieve.

SCRUM Process model

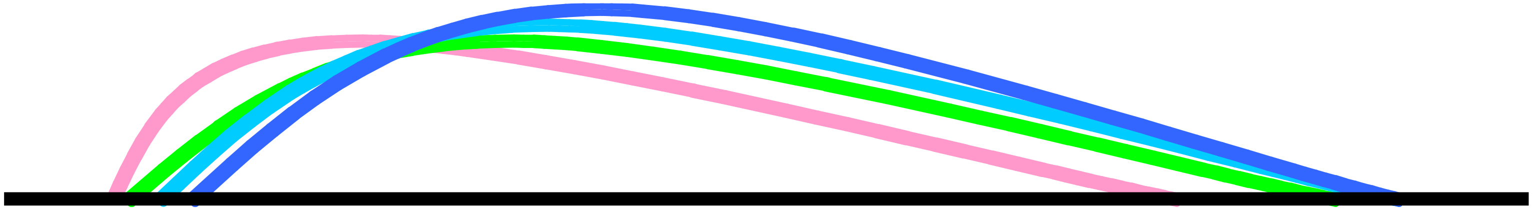
- Scrum is a process framework within which you can employ various processes and techniques.
- Scrum makes clear the relative efficacy of your product management and development practices so that you can improve.
- The Scrum framework consists of
  - Scrum Teams and their associated roles, events, artifacts, and rules.
  - Each component within the framework serves a specific purpose and is essential to Scrum's success and usage.
- The rules of Scrum bind together the events, roles, and artifacts, governing the relationships and interaction between them.

Sequential vs. Overlap

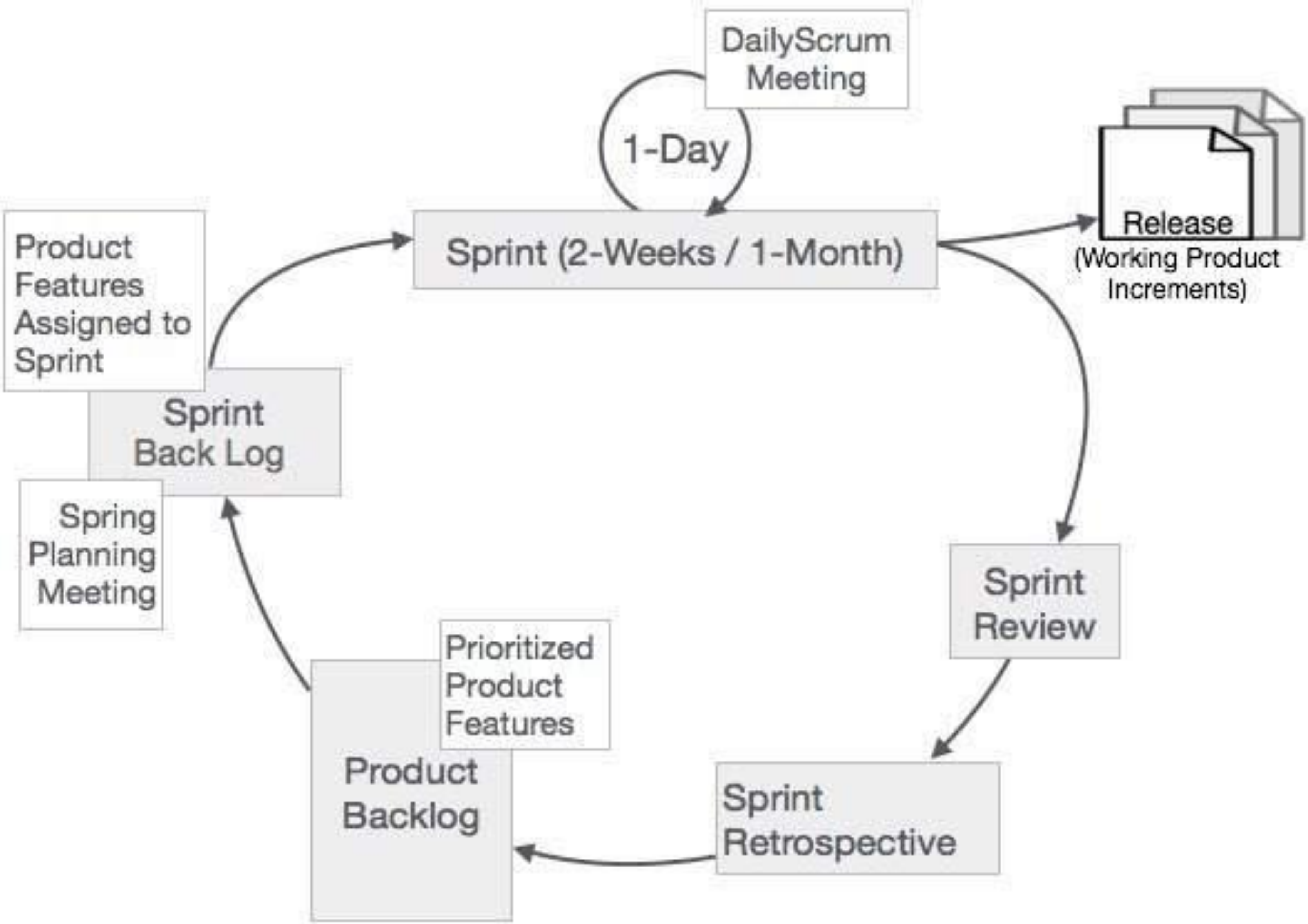
Requirements | Design | Code | Test

Rather than doing all of one thing at a time...

...Scrum teams do a little of everything all the time



Scrum Framework


DailyScrum Meeting
1-Day
Sprint (2-Weeks / 1-Month)
Release
(Working Product Increments)
Product Features Assigned to Sprint
Sprint Back Log
Spring Planning Meeting
Sprint Review
Prioritized Product Features
Product Backlog
Sprint Retrospective

Scrum Framework

Roles

- Product owner
- Scrum Master
- Team

Ceremonies

- Sprint planning
- Sprint review
- Sprint retrospective
- Daily scrum meeting

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

Scrum Roles

- **Product Owner**
  - Possibly a Product Manager or Project Sponsor
  - Decides features, release date, prioritization,

- **Scrum Master**
  - Typically a Project Manager or Team Leader
  - Responsible for enacting Scrum values and practices
  - Remove impediments / politics, keeps everyone productive

- **Project Team**
  - 5-10 members; Teams are self-organizing
  - Cross-functional: QA, Programmers, UI Designers, etc.
  - Membership should change only between sprints

Sprint in SCRUM

- The heart of Scrum is a Sprint,
  - a time-box of two weeks or one month during which a potentially releasable product increment is created.
- A new Sprint starts immediately after the conclusion of the previous Sprint.
- Sprints consist of the **Sprint planning, daily scrums, the development work, the Sprint review, and the Sprint retrospective**.
- In Sprint planning, the work to be performed in the Sprint is planned collaboratively by the Scrum Team.
- The Daily Scrum Meeting is a 15-minute time-boxed event for the Scrum Team to synchronize the activities and create a plan for that day.

Sprint in SCRUM

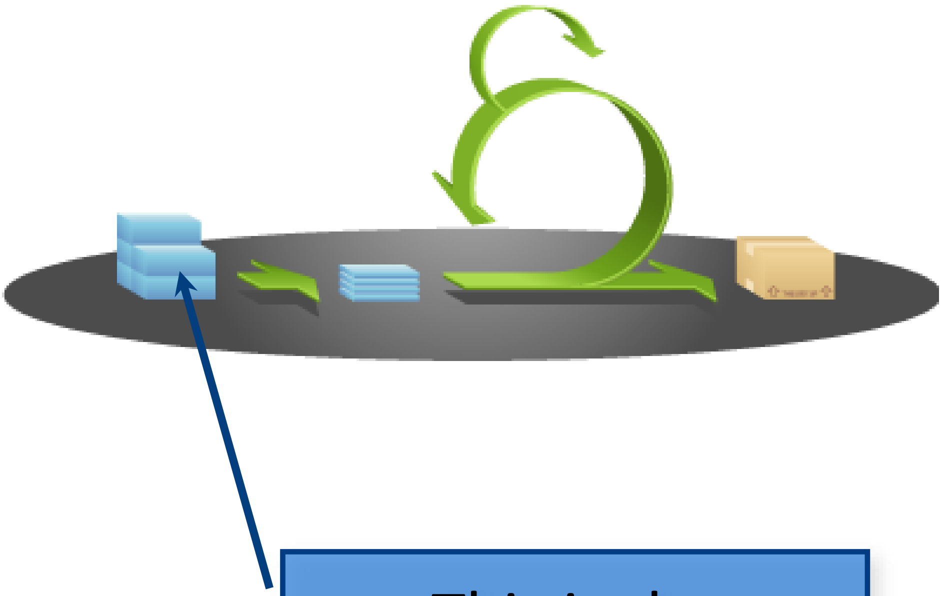
- A Sprint Review is held at the end of the Sprint to inspect the Increment and make changes to the Product Backlog, if needed.
- The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning.
- In this meeting, the Scrum Team is to inspect itself and create a plan for improvements to be enacted during the subsequent Sprint.

Scrum's Artifacts

- Scrum has remarkably few artifacts
  - Product Backlog
  - Sprint Backlog
  - Burndown Charts

- Can be managed using just an Excel spreadsheet
  - More advanced / complicated tools exist:
    - Expensive
    - Web-based – no good for Scrum Master/project manager who travels
    - Still under development

Product Backlog



This is the product backlog

- The requirements
- A list of all desired work on project
- Ideally expressed as a list of user stories along with "story points", such that each item has value to users or customers of the product
- Prioritized by the product owner
- Reprioritized at start of each sprint

User Stories

- Instead of Use Cases, Agile project owners do "user stories"
  - Who (user role) – Is this a customer, employee, admin, etc.?
  - What (goal) – What functionality must be achieved/developed?
  - Why (reason) – Why does user want to accomplish this goal?

  As a [user role], I want to [goal], so I can [reason].

- Example:
  - "As a user, I want to log in, so I can access subscriber content."
- story points: Rating of effort needed to implement this story
  - common scales: 1-10, shirt sizes (XS, S, M, L, XL), etc.

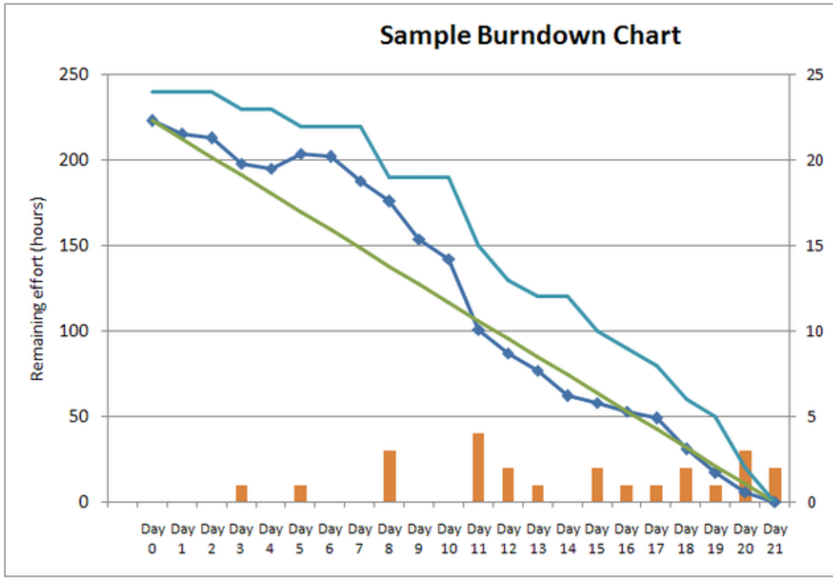
Sprint Backlog

- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily
- Any team member can add, delete change sprint backlog
- Work for the sprint emerges
- If work is unclear, define a sprint backlog item with a larger amount of time and break it down later
- Update work remaining as more becomes known

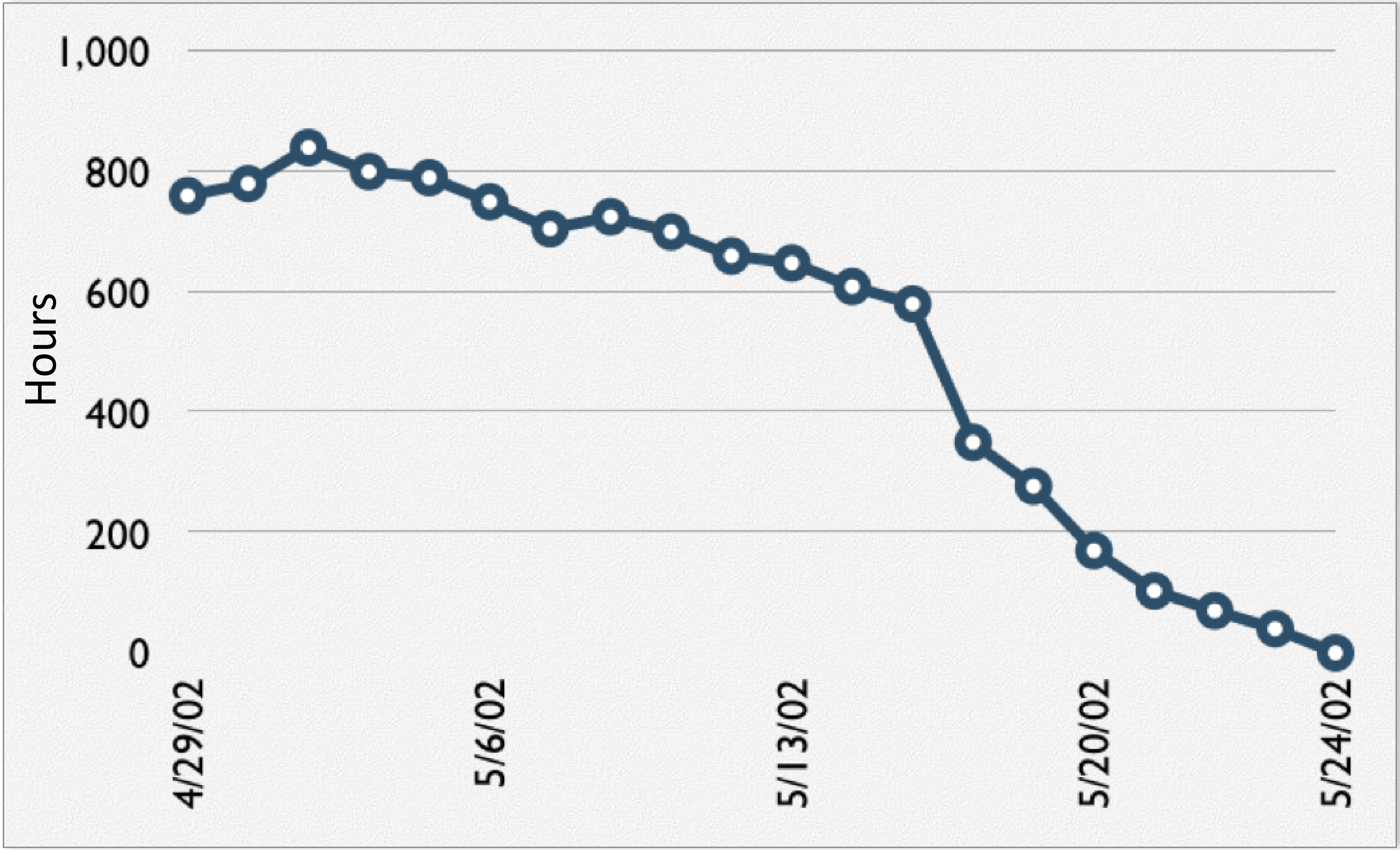
Sprint Burndown Chart

- A display of what work has been completed and what is left to complete
  - one for each developer or work item
  - updated every day
  - (make best guess about hours/points completed each day)

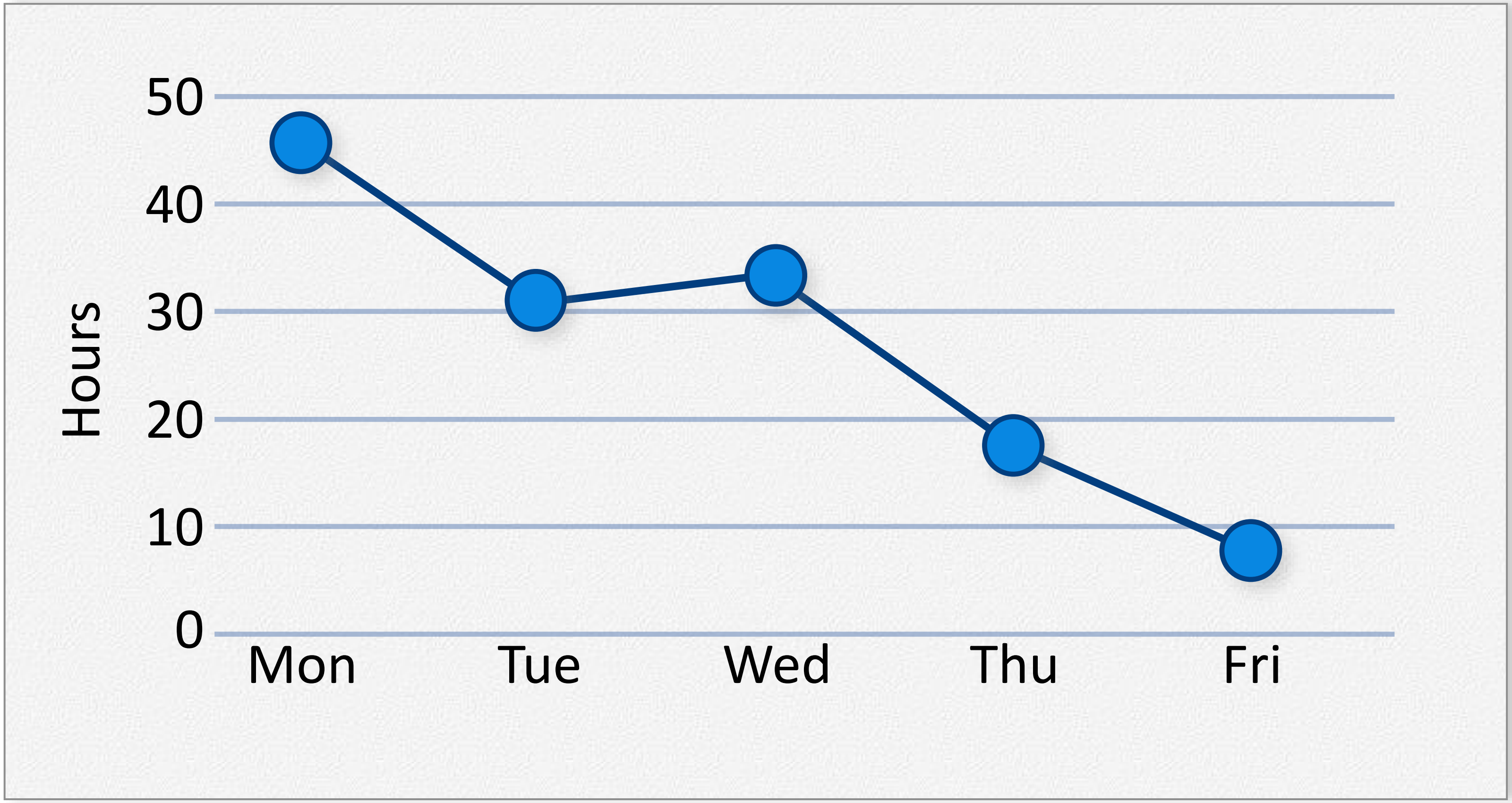
- *variation:* Release burndown chart
  - shows overall progress
  - updated at end of each sprint



Sample Burndown Chart



Tasks	Mon	Tue	Wed	Thu	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	7	
Test the middle tier	8	16	16	11	8
Write online help	12				



Project characteristics not suited to development using agile models

- **Stable requirements:** Conventional development models are more suited to use in projects characterized by stable requirements. For such projects, it is known that few changes, if at all, will occur.
- **Mission critical or safety critical systems:** In the development of such systems, the traditional SDLC models are usually preferred to ensure reliability.

Comparison of Different Life Cycle Models

- Iterative waterfall model
  - most widely used model.
  - But, suitable only for well-understood problems.
  - Not suitable for development of very large projects and projects that suffer from large number of risks.

Comparison of Different Life Cycle Models

- Prototype model is suitable for projects not well understood:
  - user requirements
  - technical aspects
  - all the risks can be identified before the project starts.
  - This model is especially popular for development of the user interface part of projects.

Comparison of Different Life Cycle Models (CONT.)

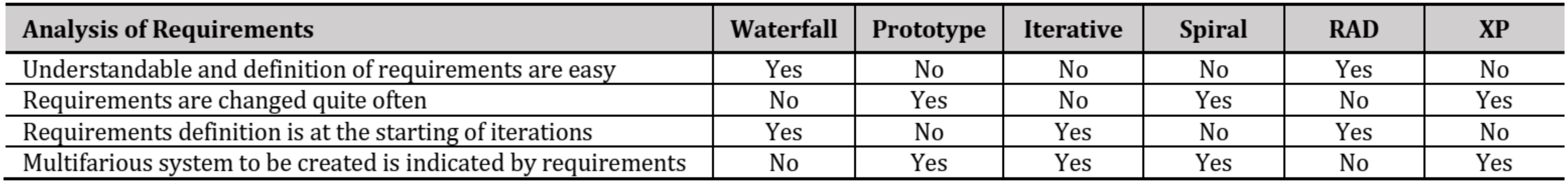
- Evolutionary model is suitable for large problems:
  - can be decomposed into a set of modules that can be incrementally implemented,
  - incremental delivery of the system is acceptable to the customer.
  - this model can only be used if incremental delivery of the system is acceptable to the customer.

Comparison of Different Life Cycle Models (CONT.)

- The spiral model:
  - suitable for development of technically challenging software products that are subject to several kinds of risks.
  - Flexibility and risk handling are inherently built into this model.
  - suitable for development of technically challenging and large software that are prone to several kinds of risks that are difficult to anticipate at the start of the project.
  - this model is much more complex than the other models

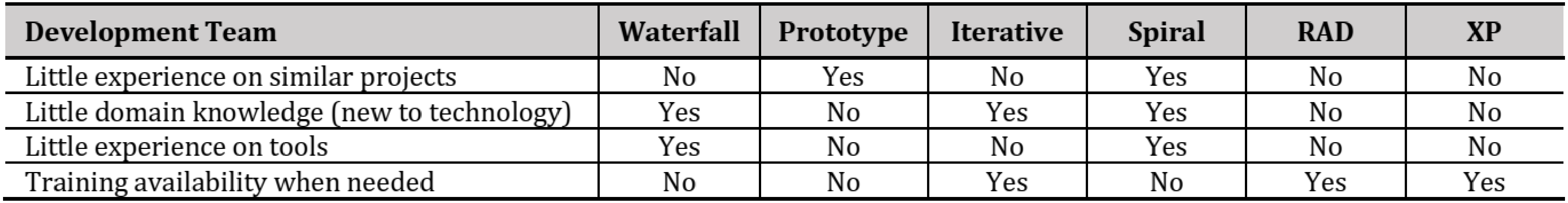
How to select a life cycle model

Table 3: Analysis of requirement [13].



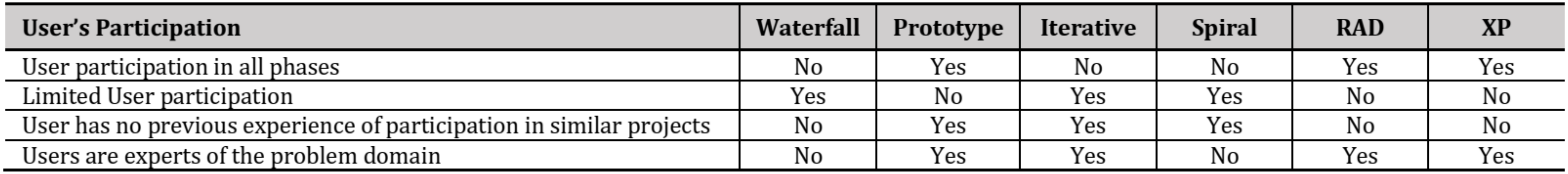
Analysis of Requirements	Waterfall	Prototype	Iterative	Spiral	RAD	XP
Understandable and definition of requirements are easy	Yes	No	No	No	Yes	No
Requirements are changed quite often	No	Yes	No	Yes	No	Yes
Requirements definition is at the starting of iterations	Yes	No	Yes	No	Yes	No
Multifarious system to be created is indicated by requirements	No	Yes	Yes	Yes	No	Yes

Table 4: Based on status of development team [13].



Development Team	Waterfall	Prototype	Iterative	Spiral	RAD	XP
Little experience on similar projects	No	Yes	No	Yes	No	No
Little domain knowledge (new to technology)	Yes	No	Yes	Yes	No	No
Little experience on tools	Yes	No	No	Yes	No	No
Training availability when needed	No	No	Yes	No	Yes	Yes

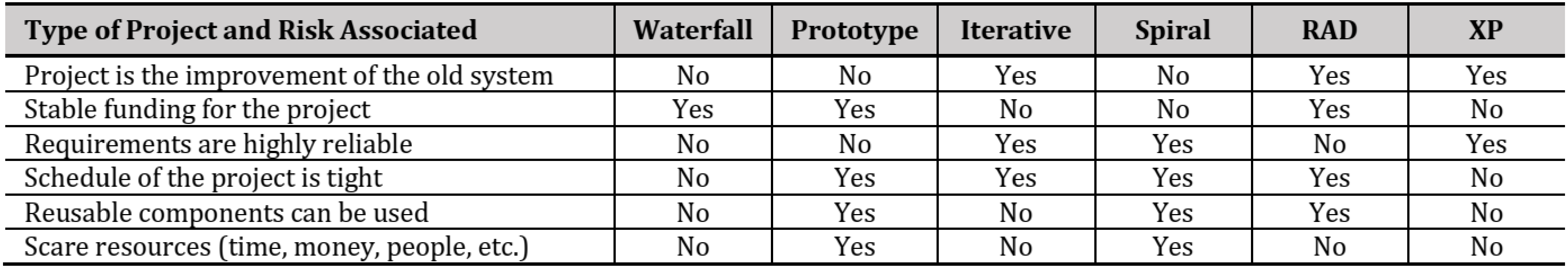
Table 5: Based on user's participation [13].



User's Participation	Waterfall	Prototype	Iterative	Spiral	RAD	XP
User participation in all phases	No	Yes	No	No	Yes	Yes
Limited User participation	Yes	No	Yes	Yes	No	No
User has no previous experience of participation in similar projects	No	Yes	Yes	Yes	No	No
Users are experts of the problem domain	No	Yes	Yes	No	Yes	Yes

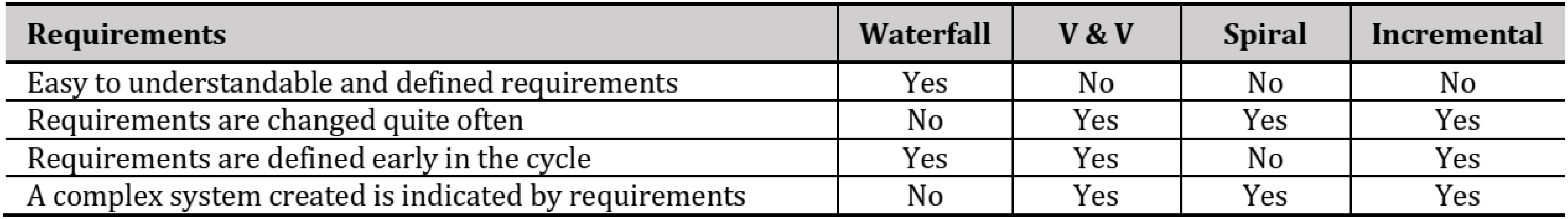
CRITERIA FORCHOOSING THERIGHTSOFTWAREDEVELOPMENTLIFE CYCLEMETHOD FOR THESUCCESS OFSOFTWAREPROJECT
AMAEFULE ANGELA ADANNA, and OGWUELEKA FRANCISCA NONYELUM, JOURNAL OF INNOVATION IN COMPUTING | ISSN: 2667-0194
VOL. 1 | ISSUE 1 | JULY 2020 | PAGES 16-26

How to select a life cycle model



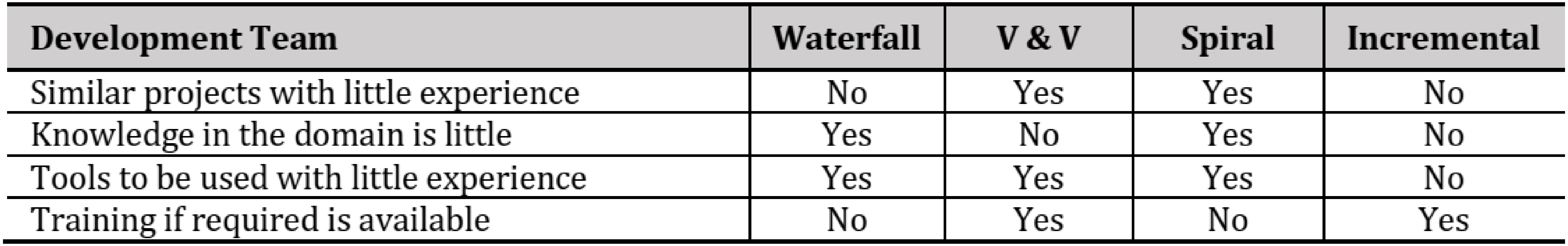
Type of Project and Risk Associated	Waterfall	Prototype	Iterative	Spiral	RAD	XP
Project is the improvement of the old system	No	No	Yes	No	Yes	Yes
Stable funding for the project	Yes	Yes	No	No	Yes	No
Requirements are highly reliable	No	No	Yes	Yes	No	Yes
Schedule of the project is tight	No	Yes	Yes	Yes	Yes	No
Reusable components can be used	No	Yes	No	Yes	Yes	No
Scare resources (time, money, people, etc.)	No	Yes	No	Yes	No	No

Table 7: Characteristics of requirements [11].



Requirements	Waterfall	V & V	Spiral	Incremental
Easy to understandable and defined requirements	Yes	No	No	No
Requirements are changed quite often	No	Yes	Yes	Yes
Requirements are defined early in the cycle	Yes	Yes	No	Yes
A complex system created is indicated by requirements	No	Yes	Yes	Yes

Table 8: Status of development team [11].



Development Team	Waterfall	V & V	Spiral	Incremental
Similar projects with little experience	No	Yes	Yes	No
Knowledge in the domain is little	Yes	No	Yes	No
Tools to be used with little experience	Yes	Yes	Yes	No
Training if required is available	No	Yes	No	Yes

How to select a life cycle model

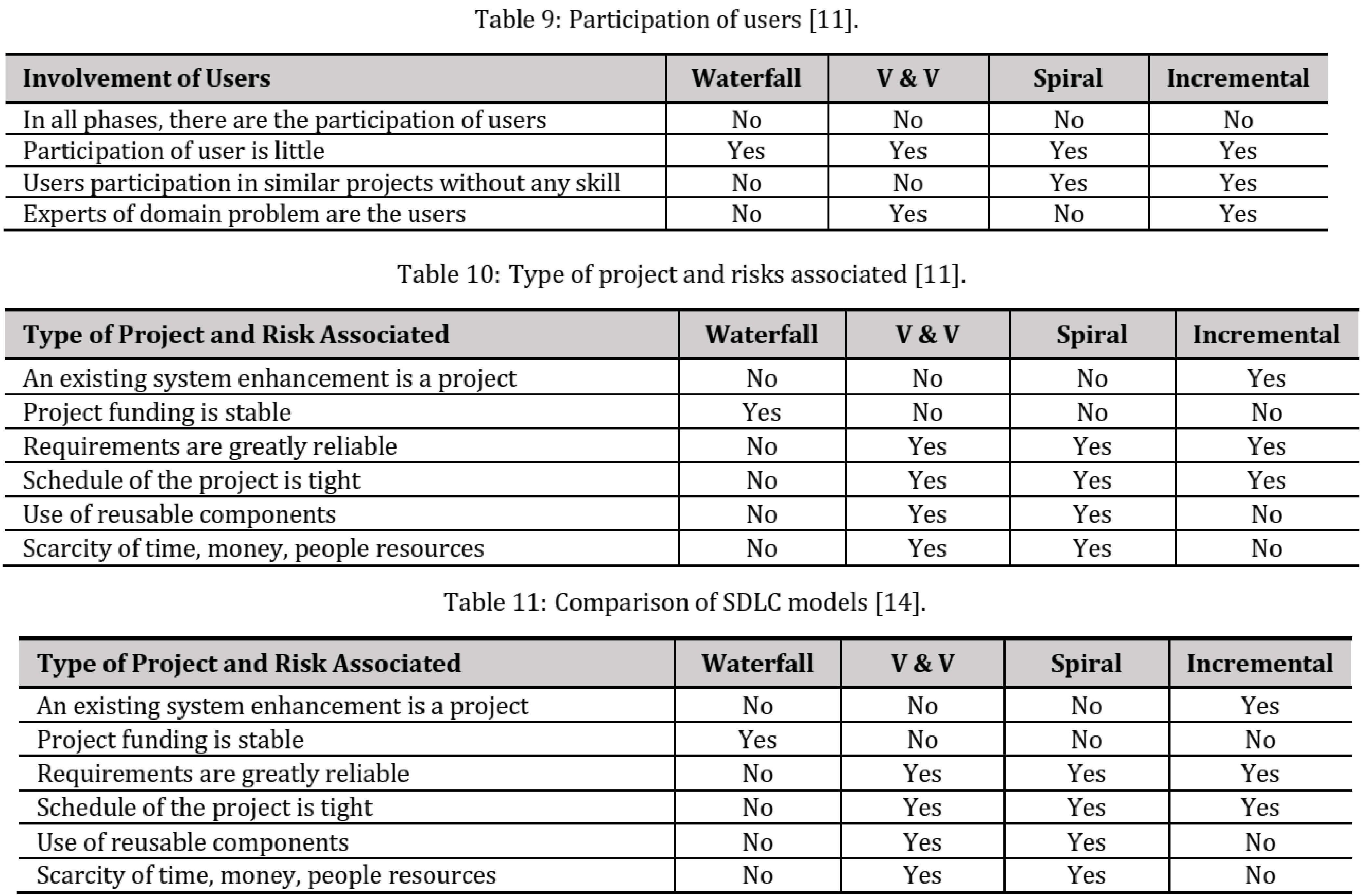


Table 9: Participation of users [11].

Involvement of Users	Waterfall	V & V	Spiral	Incremental
In all phases, there are the participation of users	No	No	No	No
Participation of user is little	Yes	Yes	Yes	Yes
Users participation in similar projects without any skill	No	No	Yes	Yes
Experts of domain problem are the users	No	Yes	No	Yes

Table 10: Type of project and risks associated [11].

Type of Project and Risk Associated	Waterfall	V & V	Spiral	Incremental
An existing system enhancement is a project	No	No	No	Yes
Project funding is stable	Yes	No	No	No
Requirements are greatly reliable	No	Yes	Yes	Yes
Schedule of the project is tight	No	Yes	Yes	Yes
Use of reusable components	No	Yes	Yes	No
Scarcity of time, money, people resources	No	Yes	Yes	No

Table 11: Comparison of SDLC models [14].

Type of Project and Risk Associated	Waterfall	V & V	Spiral	Incremental
An existing system enhancement is a project	No	No	No	Yes
Project funding is stable	Yes	No	No	No
Requirements are greatly reliable	No	Yes	Yes	Yes
Schedule of the project is tight	No	Yes	Yes	Yes
Use of reusable components	No	Yes	Yes	No
Scarcity of time, money, people resources	No	Yes	Yes	No

How to select a life cycle model

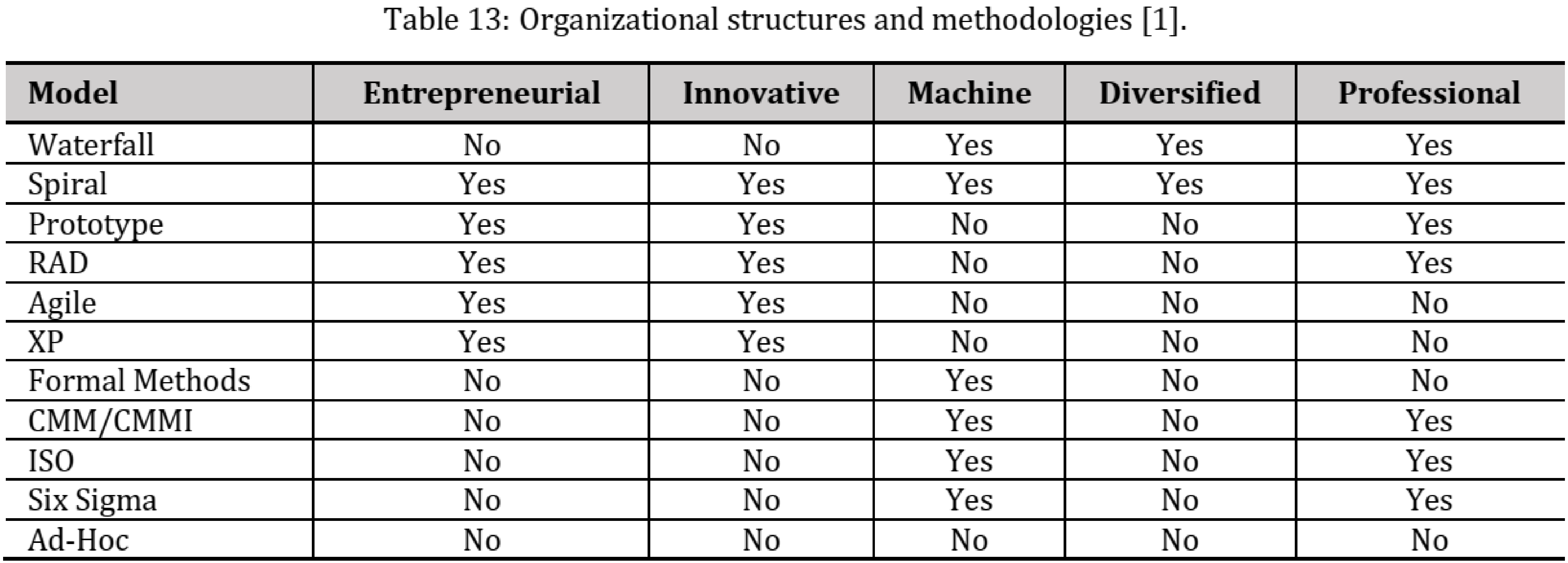


Table 13: Organizational structures and methodologies [1].

Model	Entrepreneurial	Innovative	Machine	Diversified	Professional
Waterfall	No	No	Yes	Yes	Yes
Spiral	Yes	Yes	Yes	Yes	Yes
Prototype	Yes	Yes	No	No	Yes
RAD	Yes	Yes	No	No	Yes
Agile	Yes	Yes	No	No	No
XP	Yes	Yes	No	No	No
Formal Methods	No	No	Yes	No	No
CMM/CMMI	No	No	Yes	No	Yes
ISO	No	No	Yes	No	Yes
Six Sigma	No	No	Yes	No	Yes
Ad-Hoc	No	No	No	No	No

	Software Development Models									
Criteria	**Waterfall**	**Prototype**	**Iterative**	**Spiral**	**RAD**	**XP**	**V & V**	**Incremental**	**Formal Method**	**Agile**
Simply defined and clear requirements	Y	N	N	N	Y	N	N	N	Y	
Regularly requirements change	N	Y	N	Y	N	Y	Y	Y	Y	
Early defined requirements in SDLC	Y	N	Y	N	Y	N	Y	Y	N	
Complex System due to requirements	N	Y	Y	Y	N	N	Y	Y	N	
User's experience on similar projects	N	Y	N	Y	N	N	Y	N	Y	
Less domain knowledge (new technology)	Y	N	Y	Y	N	N	N	N		
Less experience on tools to be used	Y	N	N	Y	N	N	Y	N		
Availability of training requirement	N	N	Y	N	Y	Y	Y	Y		
Users participation in all phases	N	Y	N	N	Y	Y	N	N	N	
Participation of user is little	Y	N	Y	Y	N	N	Y	Y	Y	
Users participation in similar projects without any skill	N	Y	Y	Y	N	N	N	Y		
Experts of domain problem are the users	N	Y	Y	N	Y	Y	Y	Y		
Project is the improvement of the old system	N	N	Y	N	Y	Y	N	Y		
Are the funds stable?	Y	Y	N	N	Y	N	N	N	Y	
High-reliability requirement	N	N	Y	Y	N	Y	Y	Y	Y	
Tight project schedule	N	Y	Y	Y	Y	N	Y	Y	N	
Use of reversible components	N	Y	N	Y	Y	N	Y	N	Y	
Is resource (time, money, people, etc.) scarce?	N	Y	N	Y	N	N	Y	N	N	
Entrepreneurial	N	Y		Y	Y	Y			N	Y
Innovative	N	Y		Y	Y	Y			N	Y
Machine	Y	N		Y	N	N			Y	N
Diversified	Y	N		Y	N	N			N	N
Professional	Y	Y		Y	Y	N			N	N
Total Occurrence	**Y=9 N=14**	**Y= 13 N=10**	**Y= 10 N=8**	**Y= 16 N=7**	**Y= 12 N=11**	**Y=8 N=15**	**Y=12 N=6**	**Y=10 N=8**	**Y=8 N=9**	**Y=2 N=3**

How to select a life cycle model

- **Characteristics of the software to be developed:** The choice of the life cycle model to a large extent depends on the nature of the software that is being developed.
- For small services projects, the agile model is favored.
- On the other hand, for product and embedded software development, the iterative waterfall model can be preferred.
- An evolutionary model is a suitable model for object-oriented development projects.

How to select a life cycle model

- **Characteristics of the development team:** The skill-level of the team members is a significant factor in deciding about the life cycle model to use.
- If the development team is experienced in developing similar software, then
even an embedded software can be developed using an iterative waterfall model.
- If the development team is entirely novice, then even a simple data processing application may require a prototyping model to be adopted.

How to select a life cycle model

- **Characteristics of the customer:** If the customer is not quite familiar with computers, then the requirements are likely to change frequently as it would be difficult to form complete, consistent, and unambiguous requirements.
- Thus, a prototyping model may be necessary to reduce later change requests from the customers.