SDLC

Definition

- Software engineering is the study and an application of engineering to the design, development and maintenance of software
- An engineering discipline that applies theories, methods, and tools to solve problems related to software production and maintenance
- Sub-disciplines:
  - Requirement engineering
  - Software design, construction, testing
  - Software configuration management
  - Software quality management
  - and more

Importance

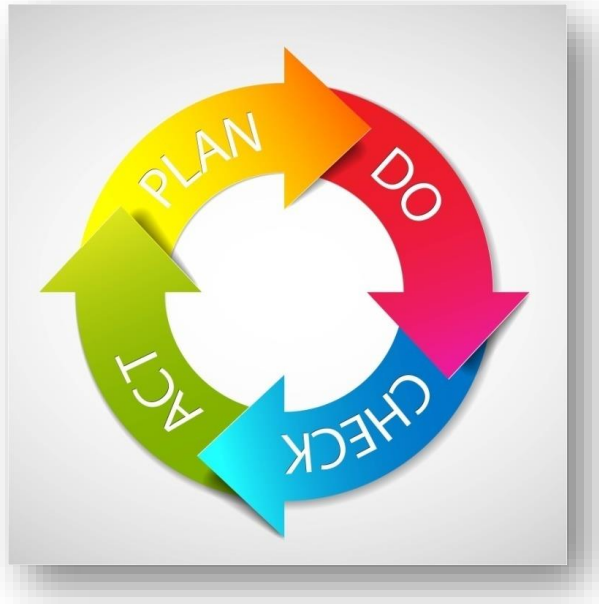
Why is Software Engineering important?

- Critical systems in finance, security, and safety rely on software
- Software mediates every aspect of our internet experience
- The economics of all developed nations are dependent on software
- There is an increasing need to develop high quality software in a cost effective manner

Software Process

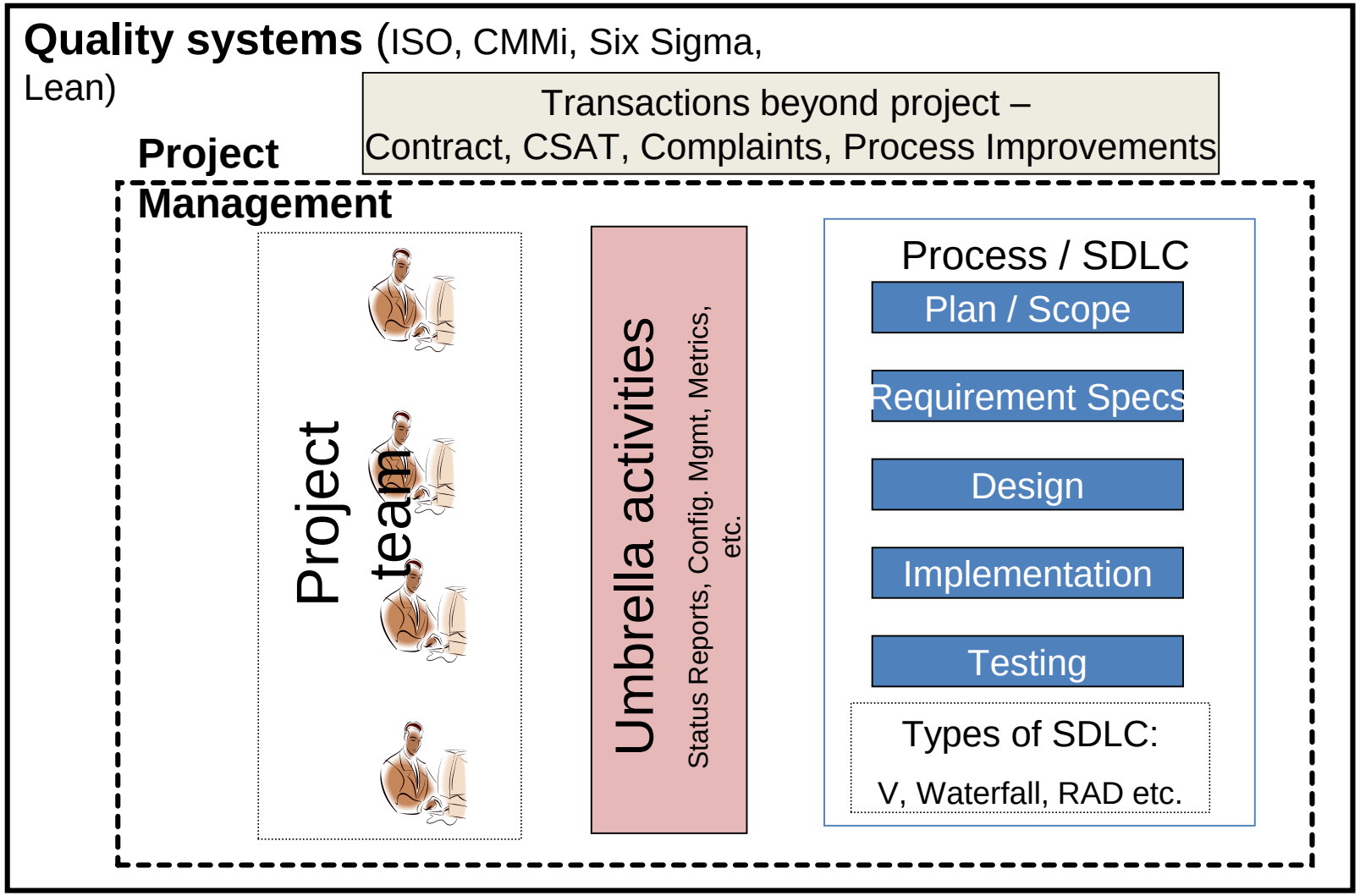
- A set of activities and their output, which results in a software product

Specification	Defines what the software should do, and its operational constraints
Design and Implementation	Designs the solutions, and produces the source code to meet the specification
Validation	Checks that the software produced is what the customer wants
Evolution	Changes made to the software that meet user's changing needs



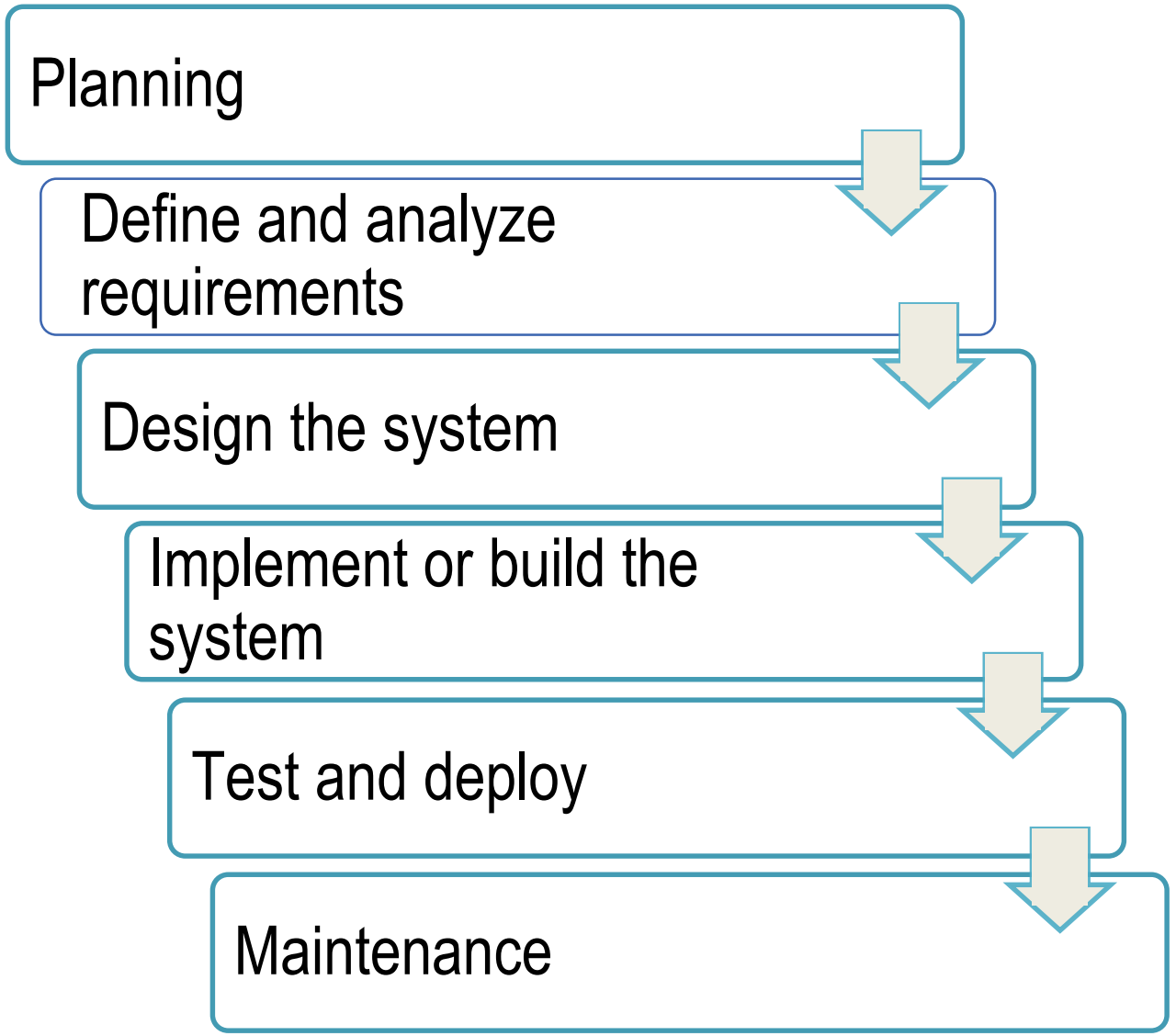
Software Process

- Quality encompasses all activities involved in a project
- Project management includes managing people, tasks, various project phases, stages, etc.
- Transactions beyond project helps to improve the overall quality of the whole system



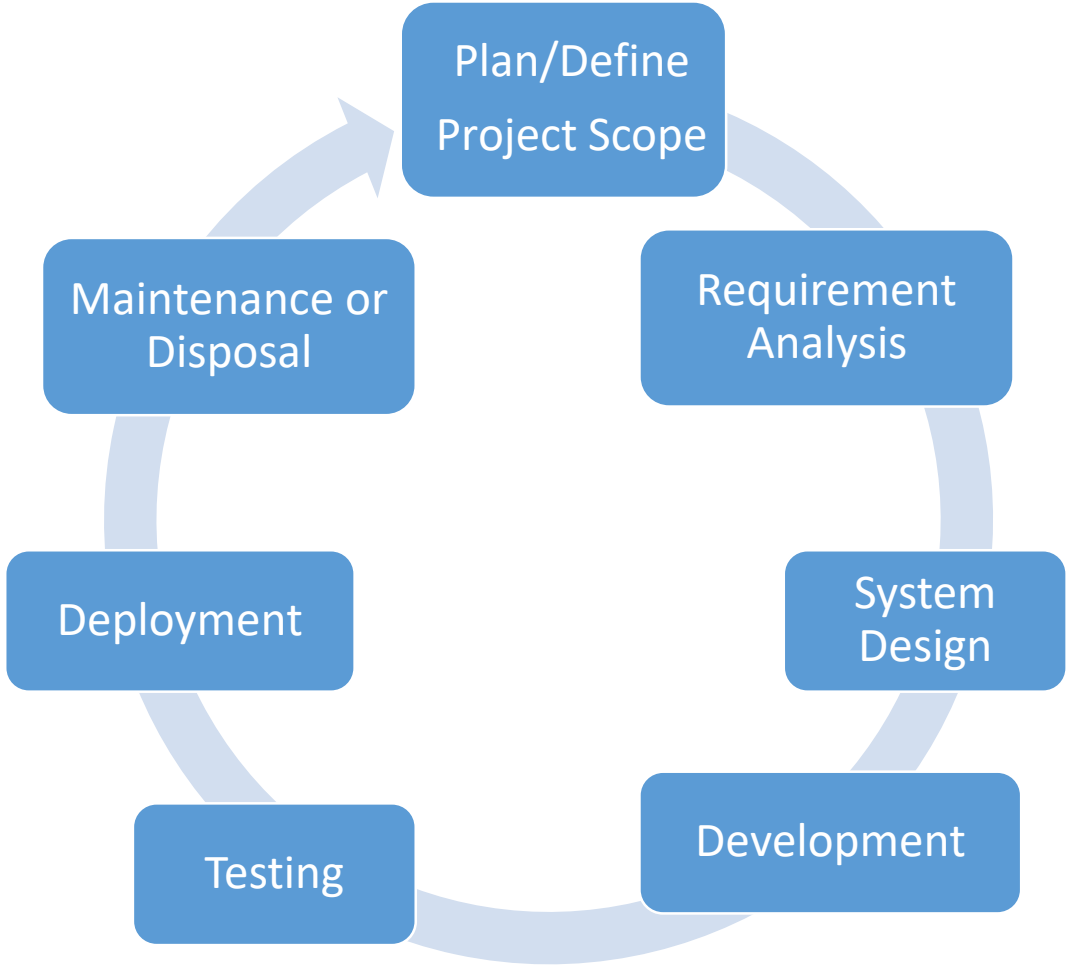
Introduction

- Software Development Life-cycle (SDLC) is a process consisting of a series of planned activities to develop or maintain software products
- Also referred to as **Software Development Process**
- At a high-level it consists of following stages:



Phases

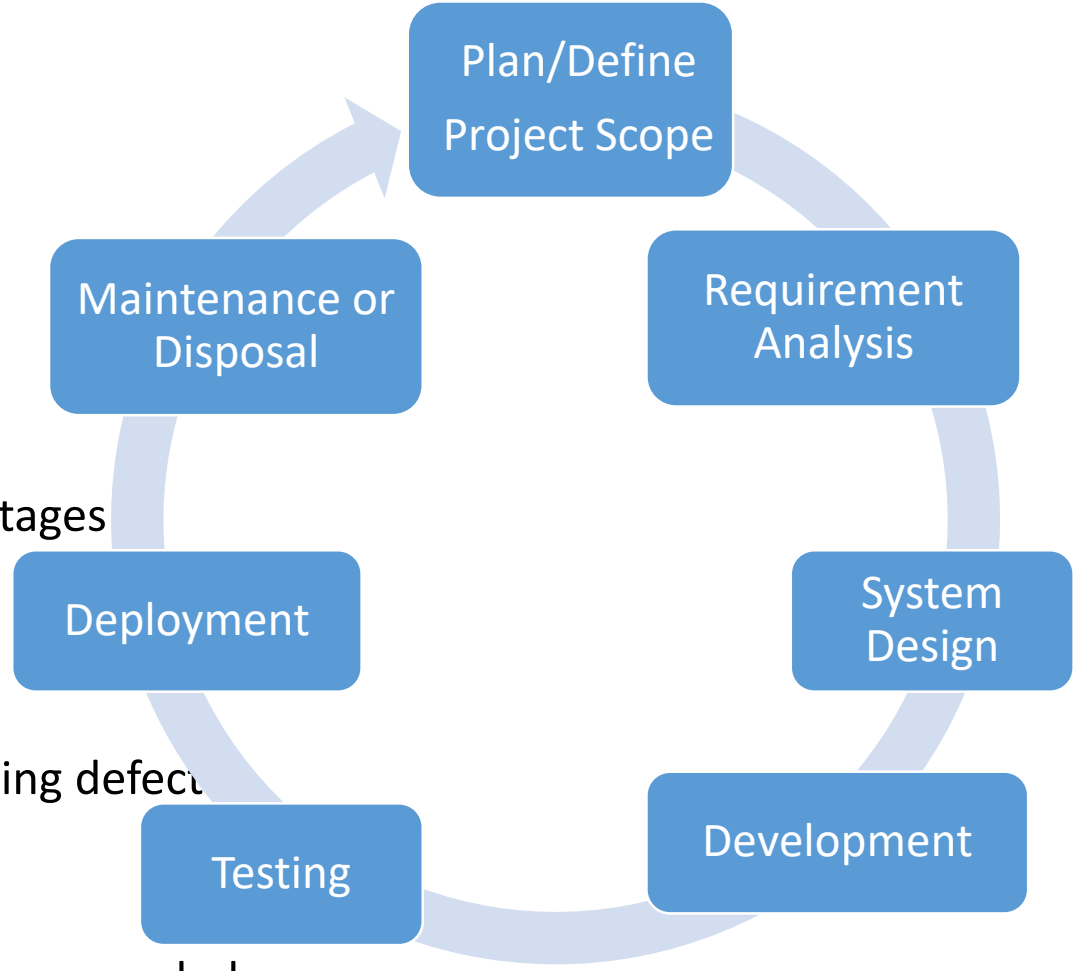
- Activities involved in Software Development Life-Cycle phases are:
  - Plan/Define Project Scope
    - Conduct preliminary analysis
    - Propose alternative solutions
    - Describe the cost and benefits
  - Requirements Analysis
    - Collect various facts
    - Study existing system
    - Analyze proposed system
    - Prepare requirement specifications
  - System Design
    - ‘How’ part of the system: Details of screens, business rules, pseudo-code, etc.
    - Test planning



Phases

- Activities involved in SDLC (contd..):
  - Development
    - Actual implementation of the system (coding)
  - Testing
    - Test system using various techniques in various stages
  - Deployment (Installation)
    - Installing the software in the final environment
  - Maintenance
    - Taking up any enhancements and fixing any existing defect

  OR
  - Disposal
    - Stop using it when it becomes obsolete or no longer needed
    - May be replaced with a new system



Phases

Initiation

Begins when a sponsor identifies a need or an opportunity. Concept Proposal is created.

System Concept Development

Defines the scope or boundary of the concepts. Includes: systems boundary document, cost benefit analysis, risk management plan and feasibility study.

Planning

Develops a Project Management Plan and other planning documents. Provides the basic for acquiring the resources needed to achieve a solution.

Requirement Analysis

Analyses user needs and develops user requirements. Creates a Detailed Functional Requirements Document.

Design

Transforms detailed requirements into complete, detailed Systems Design Documents. Focuses on how to deliver the required functionality.

Development

Converts a design into a complete information system. Includes acquiring and installing systems environment; creating and testing databases, preparing test cases procedure, preparing test files, coding, compiling, refining, programs; performing test readiness review and procurement activities.

Integration and Testing

Demonstrates that developed system conforms to requirements as specified in the functional requirements document. Conducted by Quality Assurance staff and users. Produces Test Analysis Reports.

Implementation

Includes implementation preparation, implementation of the system into a production environment, and resolution of problems identified in the integration and Test Phases.

Operations and Maintenance

Describes tasks to operate and maintain information systems in a production environment includes Post-Implementation and In-Process Reviews.

Disposition

Describes end-of-system activities. Emphasis is given to proper preparation of data.

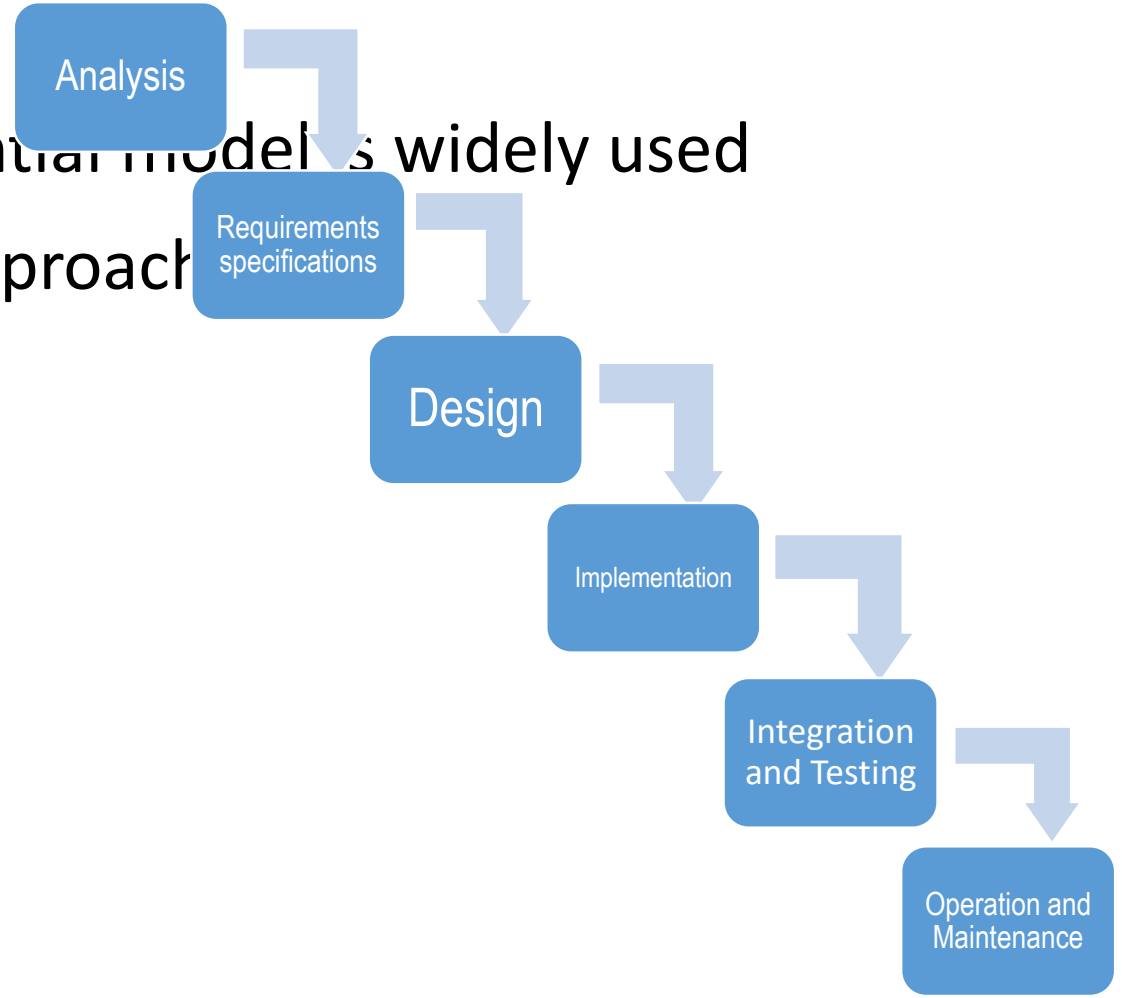
SDLC Models

Generic software process models

- Waterfall model
- Evolutionary models
  - Prototyping model
  - Spiral model
- Incremental models
  - Iterative model
  - RAD model
- Agile models
  - Extreme Programming (XP)
  - Rational Unified Process (RUP)
  - Scrum

Waterfall Model

- Classical life cycle and linear sequential model is widely used
- Suggests a systematic sequential approach
  - Conception & Initiation
  - Analysis
  - Design
  - Construction
  - Testing
  - Deployment
  - Maintenance / Support



Waterfall Model

Phases of Waterfall Model

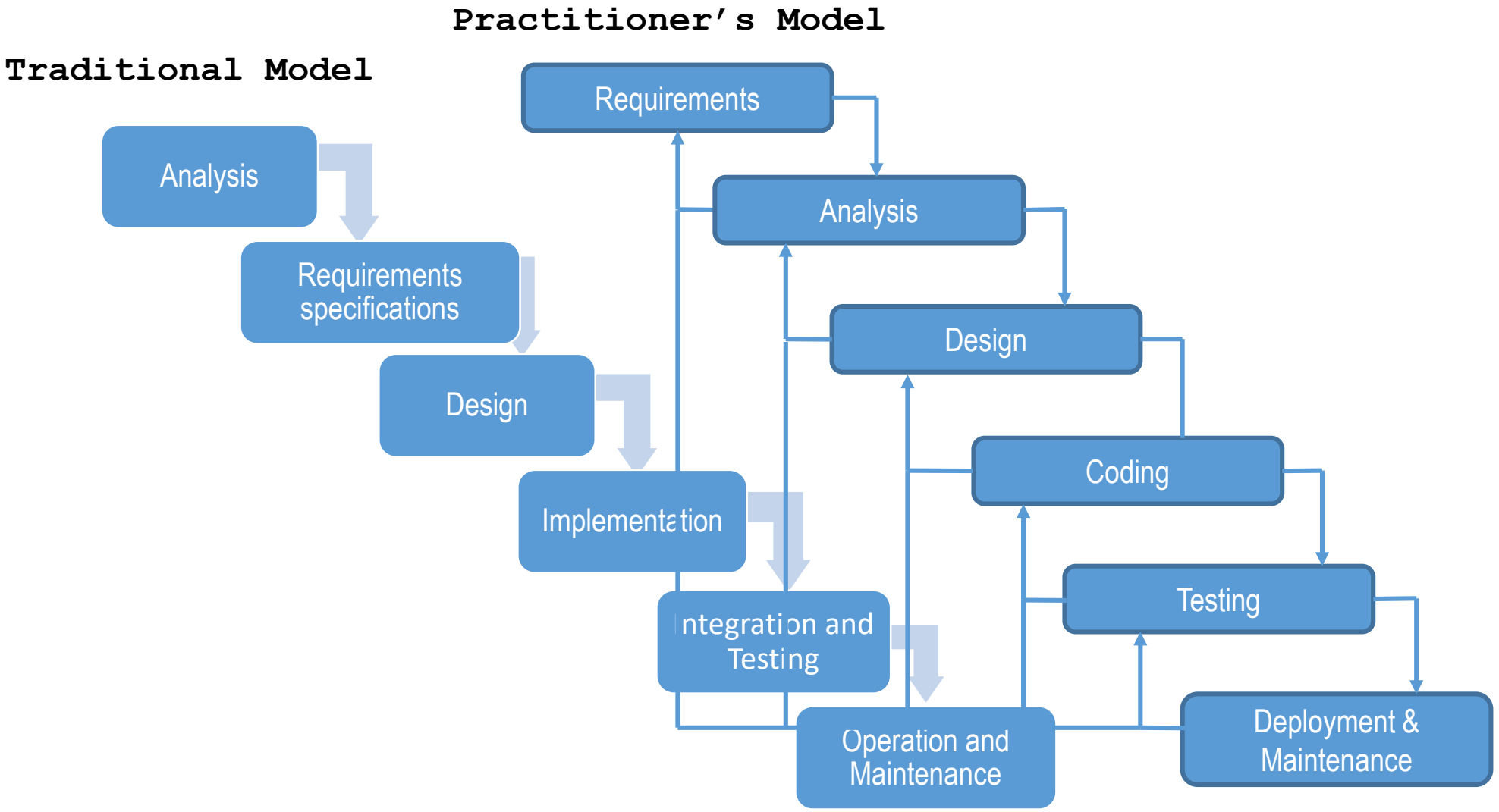
- Initial analysis
  - Specifying the problem background, business goals and success criteria
- Requirement analysis and definition
  - The goals and constraints of the system to be developed are established in consultation with system users
  - They are then defined in detail and serve as a system specification
- System and software design
  - Partitions the requirements to either hardware or software
  - Software design involves identification of necessary fundamentals of the software requirement and their relationship with those systems
- Implementation
  - Whole of software components are integrated as a set of programming units

Waterfall Model

Phases of Waterfall Model (contd …)

- Unit testing
  - Involves verifying that each unit meets its necessary requirement
- Integration and system testing
  - The individual program units are integrated and tested as a one complete system to ensure that all the necessary requirements have been met.
- Deployment/installation
  - After testing the system is installed in the final environment
- Operation and maintenance
  - Longest life cycle phase
  - Involves correcting the encountered errors which were not discovered at earlier stages
  - Also by enhancing the systems services, new requirements are discovered

Waterfall Model



Waterfall Model

Advantages

- Easy to understand and Implement
- Widely used and known
- Reinforces good habits: define-before-design
- Identifies deliverables and milestones
- Document driven
- Works well on mature products and weak teams
- Fits other engineering process models

Waterfall Model

Disadvantages

- Idealized, doesn’t match reality well
- Does not reflect iterative nature of exploratory development
- Unrealistic to expect accurate requirements early in the project
- Delays discovery of errors
- Difficult to integrate risk management
  - Difficult to accommodate change after the process is underway
  - One phase has to be competed before moving to the next phase.
- Difficult and expensive to make changes to document

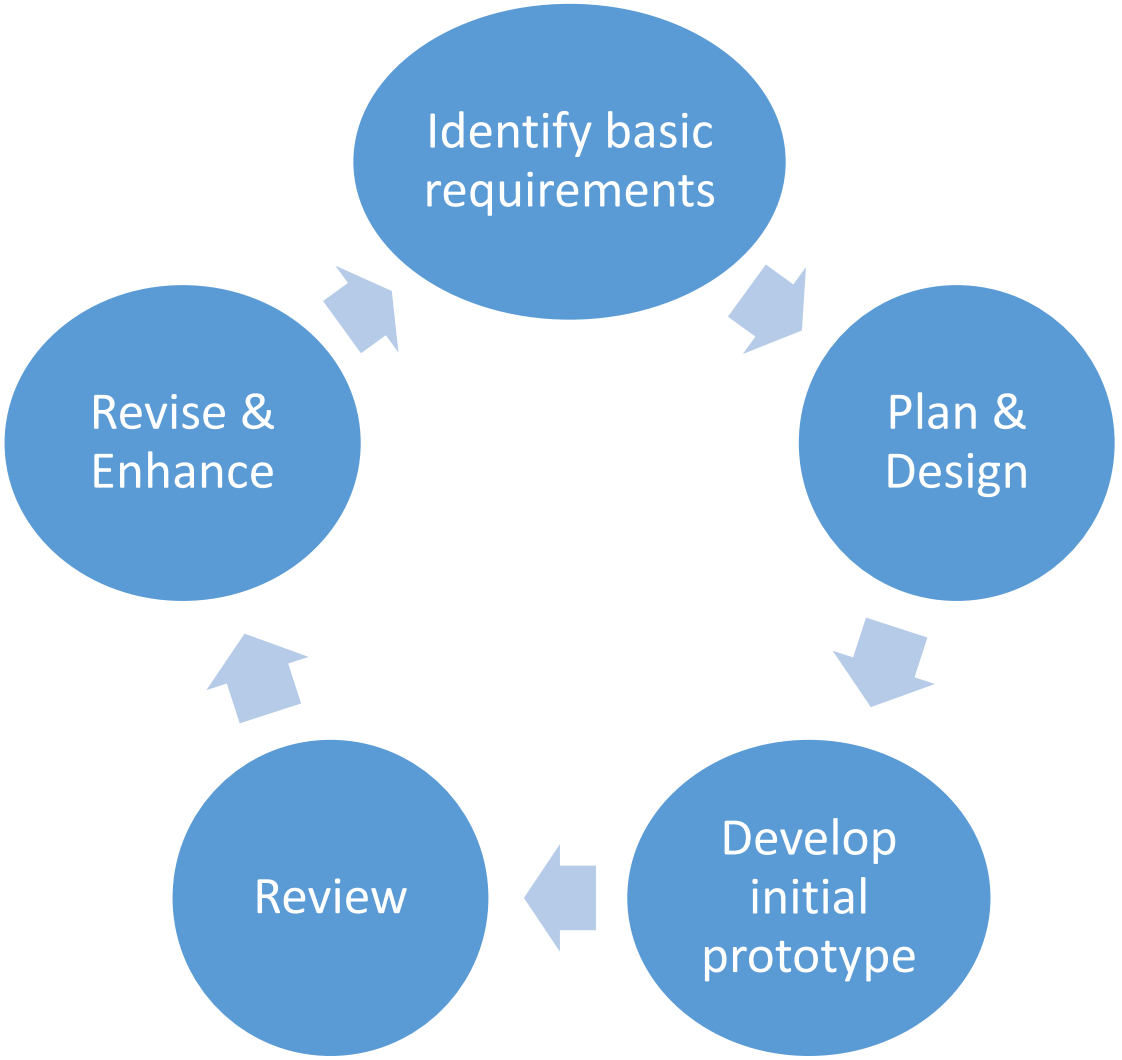
Waterfall Model

When to use?

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

Prototyping Model

- Allows users of the software to evaluate developers' proposals for the design of the final product by actually trying them out by means of a prototype of the screens
- Avoids the hardship of end users having to interpret and evaluate the design based on descriptions
- Users review and provide feedback on changes required for the prototype
- This is repeated until all requirements of system are identified



Prototyping Model

Advantages

- Users are actively involved in the developm
- Errors can be detected much earlier
- Quicker user feedback is available
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified

Prototyping Model

Drawbacks

- Leads to implementation and then repairing
- Practically this methodology may increase t
  system, as scope of system may expand bey
- Incomplete or inadequate problem analysis

Prototyping Model

When to use?

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- Short-lived demonstrations
- New, original development

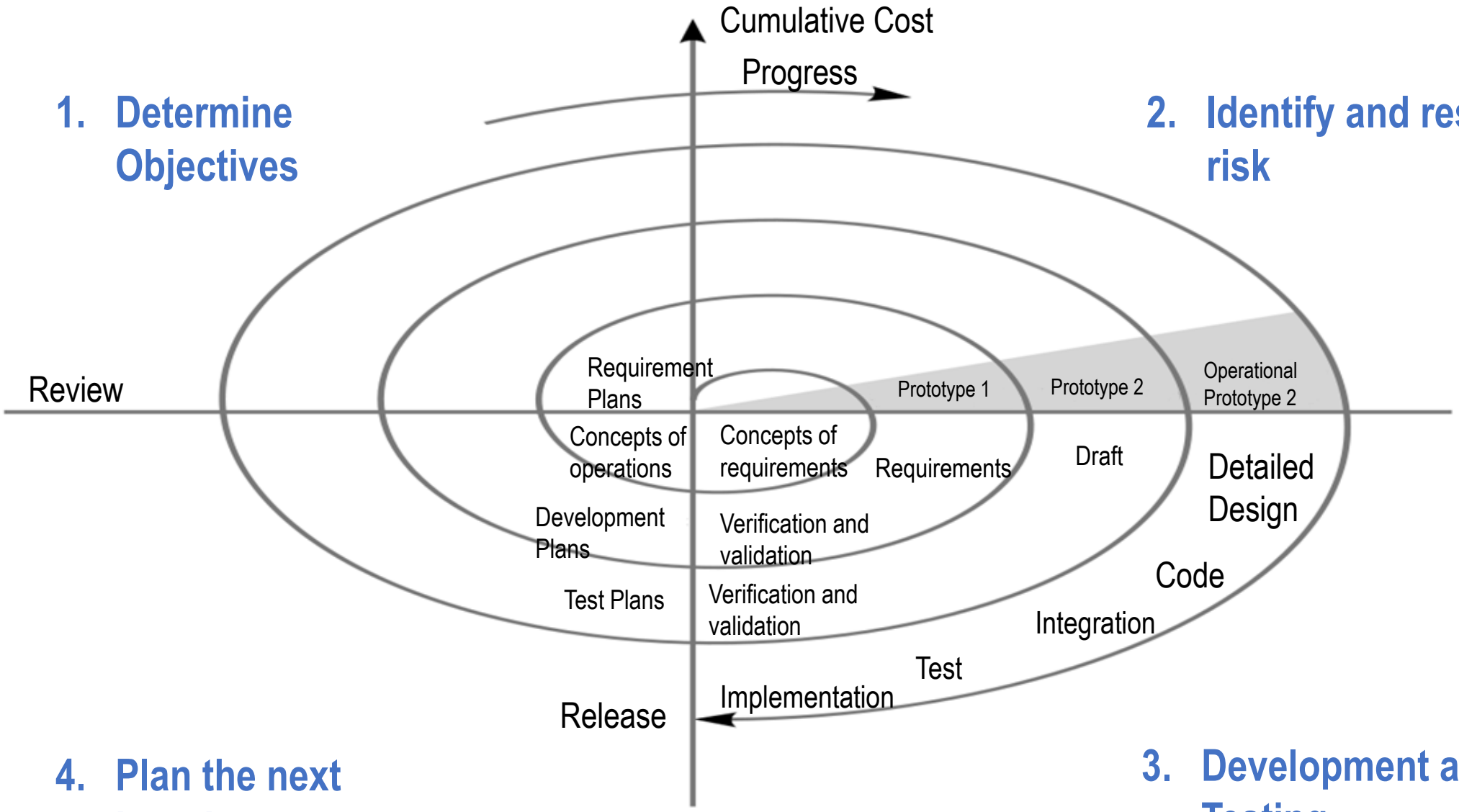
Spiral Model

- Combines the idea of iterative development with the systematic, controlled aspects of waterfall model.
- Has four phases:
  - Planning - requirements gathering phase (initial and also in subsequent spirals)
  - Risk analysis - identification of risk and alternative solutions. Creation of prototype at the end of this phase
  - Engineering - Development, testing & delivery of software
  - Evaluation - Evaluation of the software by the customer before the project continues to the next spiral

Spiral Model

1. **Determine Objectives**

2. **Identify and resolve risk**



4. **Plan the next iteration**

3. **Development and Testing**

Spiral Model

Advantages

- High amount of risk is resolved
- Software is produced at an early stage in the s
- Good for large and mission-critical projects
- Strong approval and documentation control

Spiral Model

Disadvantages

- It may be difficult to convince customers that approach is controllable
- It demands considerable risk assessment
- Can be a costly model to use
- Does not work well for small projects

Spiral Model

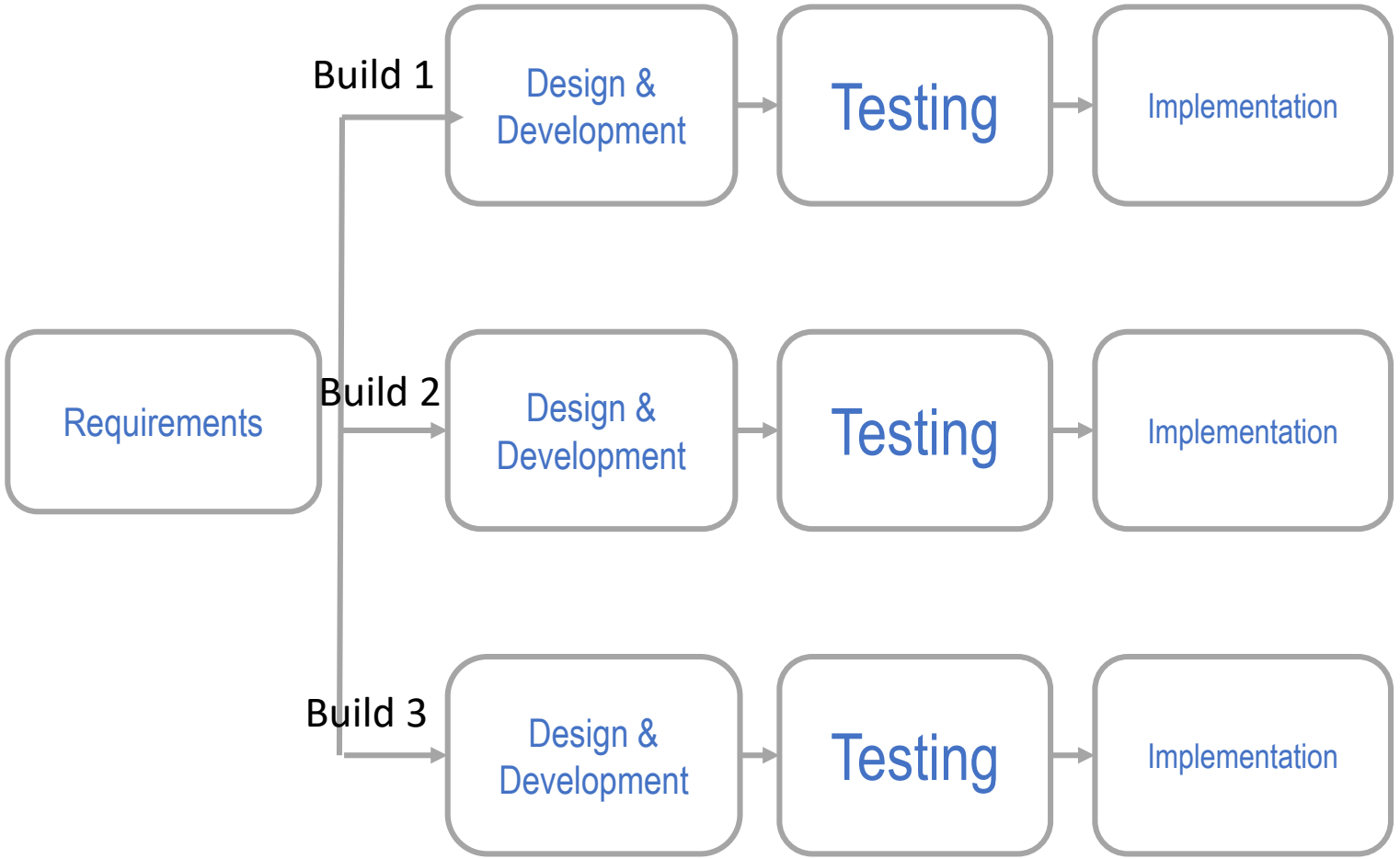
When to use?

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

Incremental Models

Iterative model

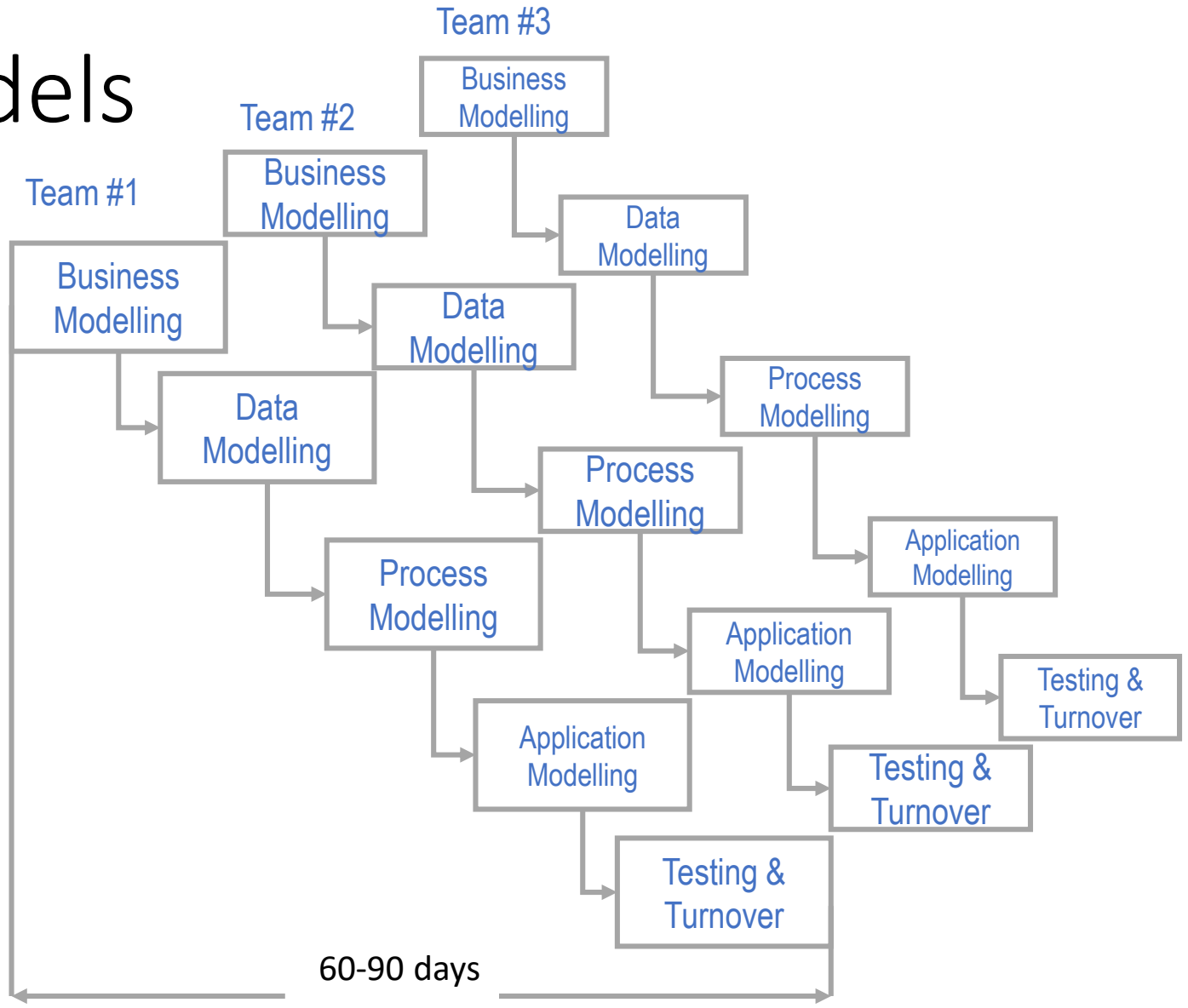
- Development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements
- The process is then repeated, producing a new version of the software for each cycle of the model
- The basic idea is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental)



Incremental Models

RAD (Rapid Application Development) model

- Components or functions are developed in parallel as if they were mini projects
- The developments are time boxed, delivered and then assembled into a working prototype
- This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements
- Phases involved in RAD model:
  - Business modeling
  - Data modeling
  - Process modeling
  - Application generation
  - Testing and turnover



Agile Model

- Driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers frequent, multiple ‘software increments’
- Adapts as changes occur
- Some agile methods:
  - Extreme Programming (XP)
    - Advocates frequent "releases" in short development cycles
    - Improves productivity and introduces checkpoints at which new customer requirements can be adopted
  - Scrum

Agile Model

Agile Team Characteristics

Small, highly motivated project team

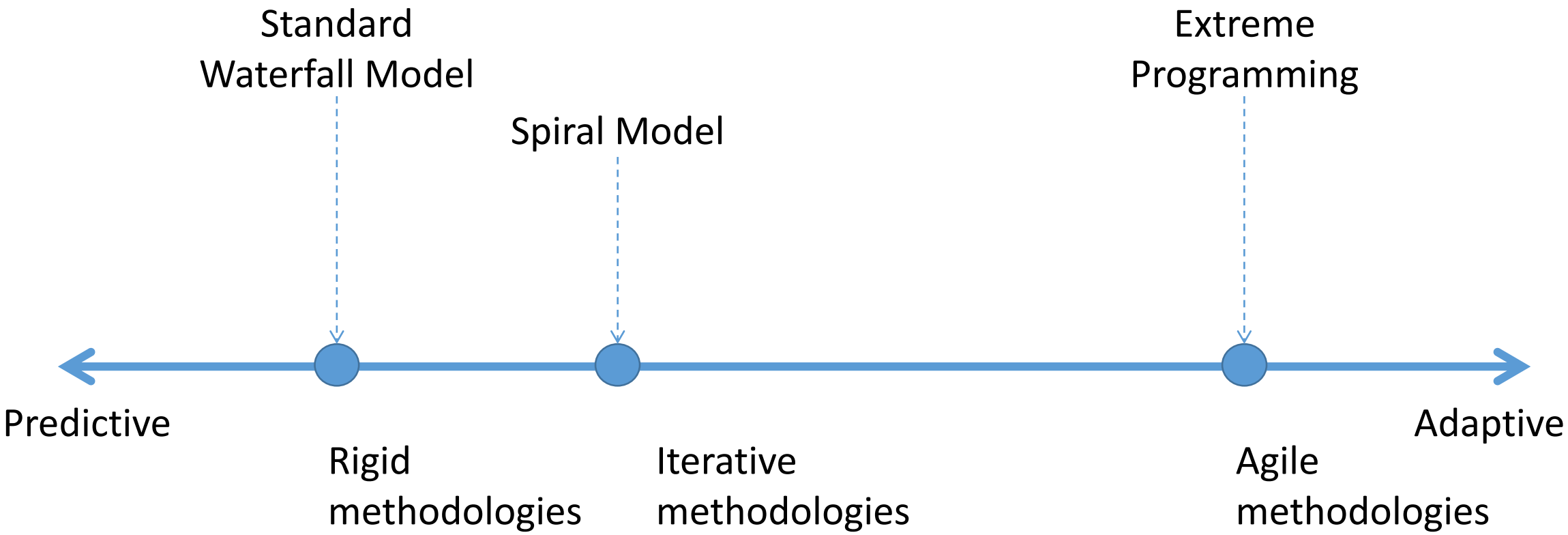
- Competence
- Common focus
- Collaboration
- Decisiveness
- Fuzzy problem-solving ability
- Mutual trust and respect
- Self-organization
- Team adopts many of the characteristics of successful software projects

- Team members must have trust with each other
- The distribution of skills must be appropriate to the problem
- Unconventional person may have to be excluded from the team, if team organization is to be maintained
- Team is “self-organizing”
- Uses elements of organizational paradigm’s random, open, and synchronous paradigms
- Significant autonomy

Agile Model

Key Principles of Agile	
(Customer) Satisfaction and delivery	through continuous working software
Welcoming change	even at the later stages of development
Deliver frequently	weekly rather than monthly
Communication is the key	between developers and business people
Environment and trust	give necessary support & trust motivated individuals
Face-to-face communication	to ensure effective & efficient communication
Software as measures of progress	working software is the primary measure of progress
Attention to details	to technical excellence and good design
Power of less	keeping it simple
Self-organizing teams	to become effective in changing circumstances

A Comparison

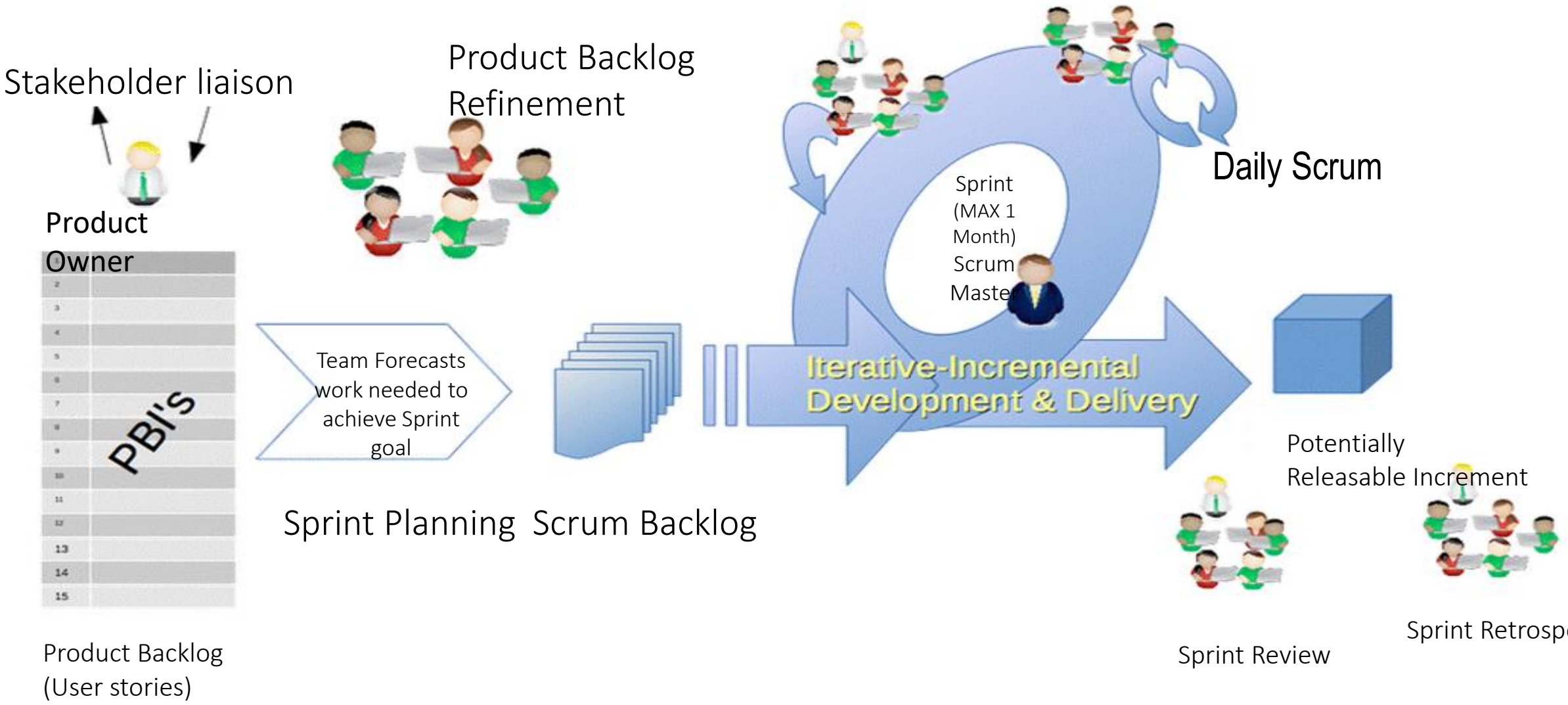


Scrum

- Small working teams are organized to “maximize communication, sharing of ideas, minimize overhead”
- The process produces frequent software increments “that can be inspected, adjusted, tested, documented, and built on”
- Scrum—distinguishing features:
  - Development work is partitioned into “packets”
  - Testing and documentation are on-going as the product is constructed
  - Scrum incorporates the following activities:
    - Requirements, Analysis, Design, Evolution, & Delivery
  - Work within activities occurs in “sprints” and is derived from a “backlog” of existing requirements

Scrum

Scrum Framework

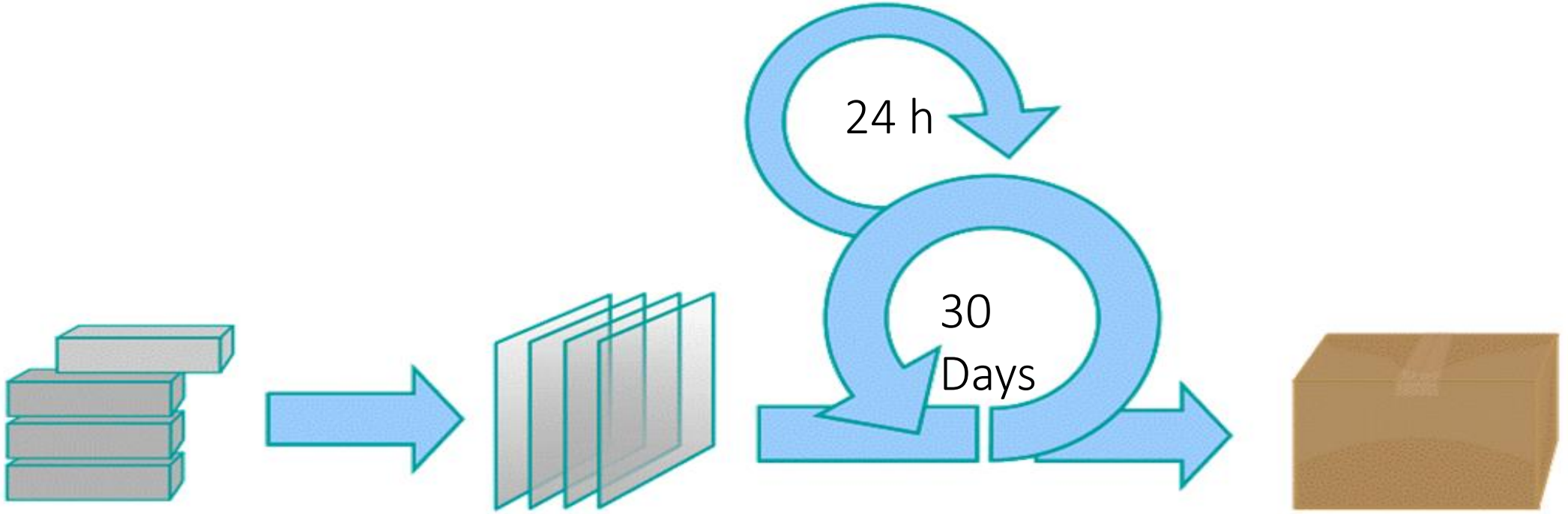


Scrum

- Product Backlog:
  - A prioritized list of project requirements or features that provides business value for the customer
  - Items can be added to the backlog at any time
  - The product manager assesses the backlog and updates priorities as required
- Sprints:
  - Consists of work units that are required to achieve a requirement
  - During the sprint, the backlog items that the sprint work units addresses are frozen (changes are not allowed)
  - The sprint allows team members to work in a short-term, but stable environment

Scrum

Scrum Process



Product Backlog

Sprint Backlog

Sprint

Working Increment of software

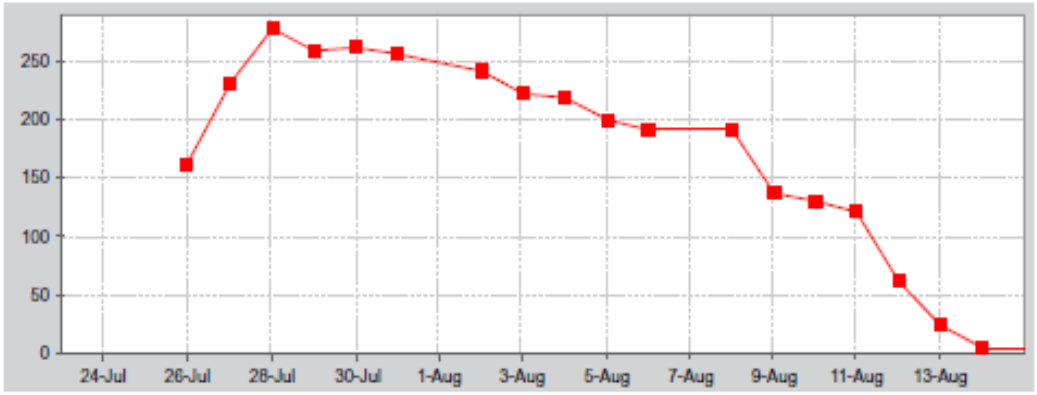
Scrum

- Scrum meetings
  - Short meetings held daily by the scrum team (also referred as "stand-up meetings")
  - Three key Q’s are asked & answered by all team members
    - What did you do since the last team meeting?
    - What obstacles are you encountering?
    - What do you plan to accomplish by the next team meeting?
  - A team leader called a “scrum master” leads the meeting & assess the responses from each person
  - Demos
  - Deliver software increment to the customer
  - Customer evaluates the functionality

Scrum

- Sprint Review:
  - The Scrum team shows what they accomplished during the sprint.
  - Typically it is a demo of the new features
- Sprint Retrospective:
  - Entire team including the Scrum master meet to discuss the lesson learnt during the just concluded sprint and also discuss about corrective measures to be taken for future sprints
- Burn down charts
  - A graphical representation of work to do against time.
  - Useful in predicting when work can get completed.
  - During sprint visual artefacts like burn down charts

    can be the big motivators for the team.



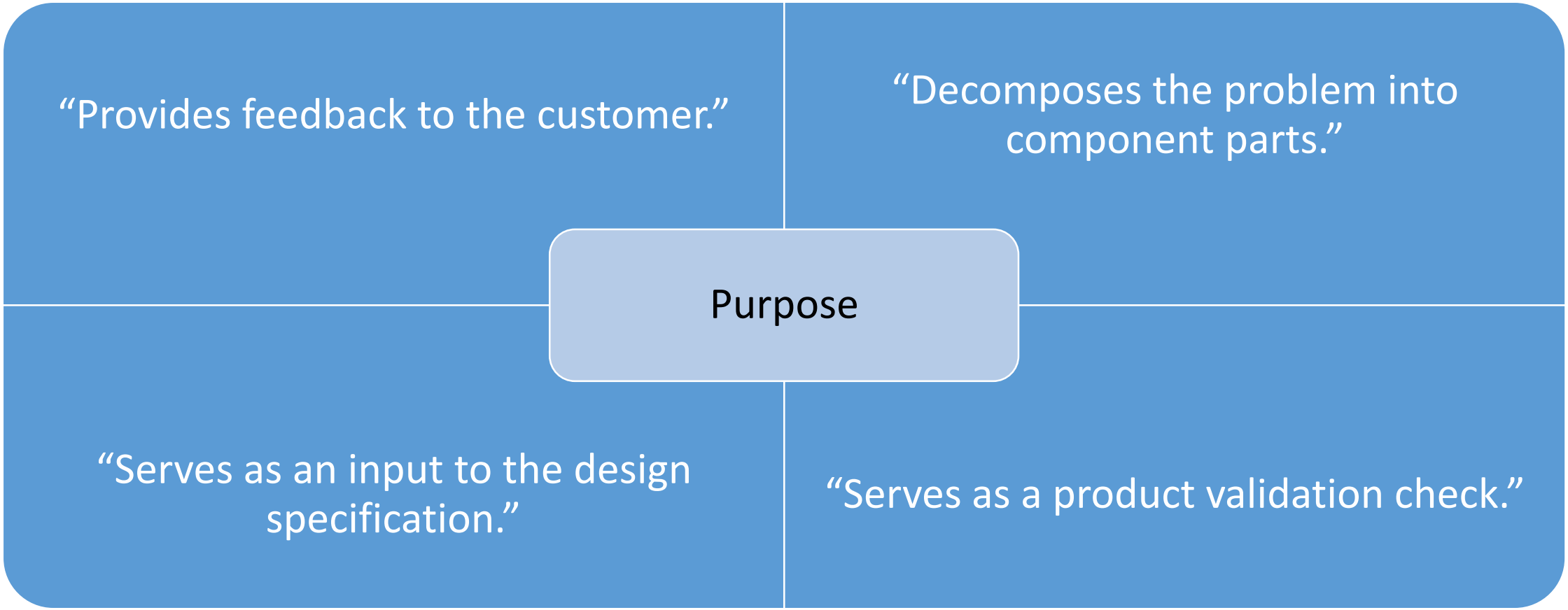
Burn Down Chart

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

What is Software Requirement Specification?

- Specifies the requirements … and the methods to be used to ensure that each requirement has been met.
- It is a document that you prepare:
  - after customer gives you their system specifications
  - before you design the system

Software Requirements Specification (SRS)



Contents

- Functional requirements
- Nonfunctional requirements
- It doesn't provide:
  - Design suggestions
  - Possible solutions to technology or business issues

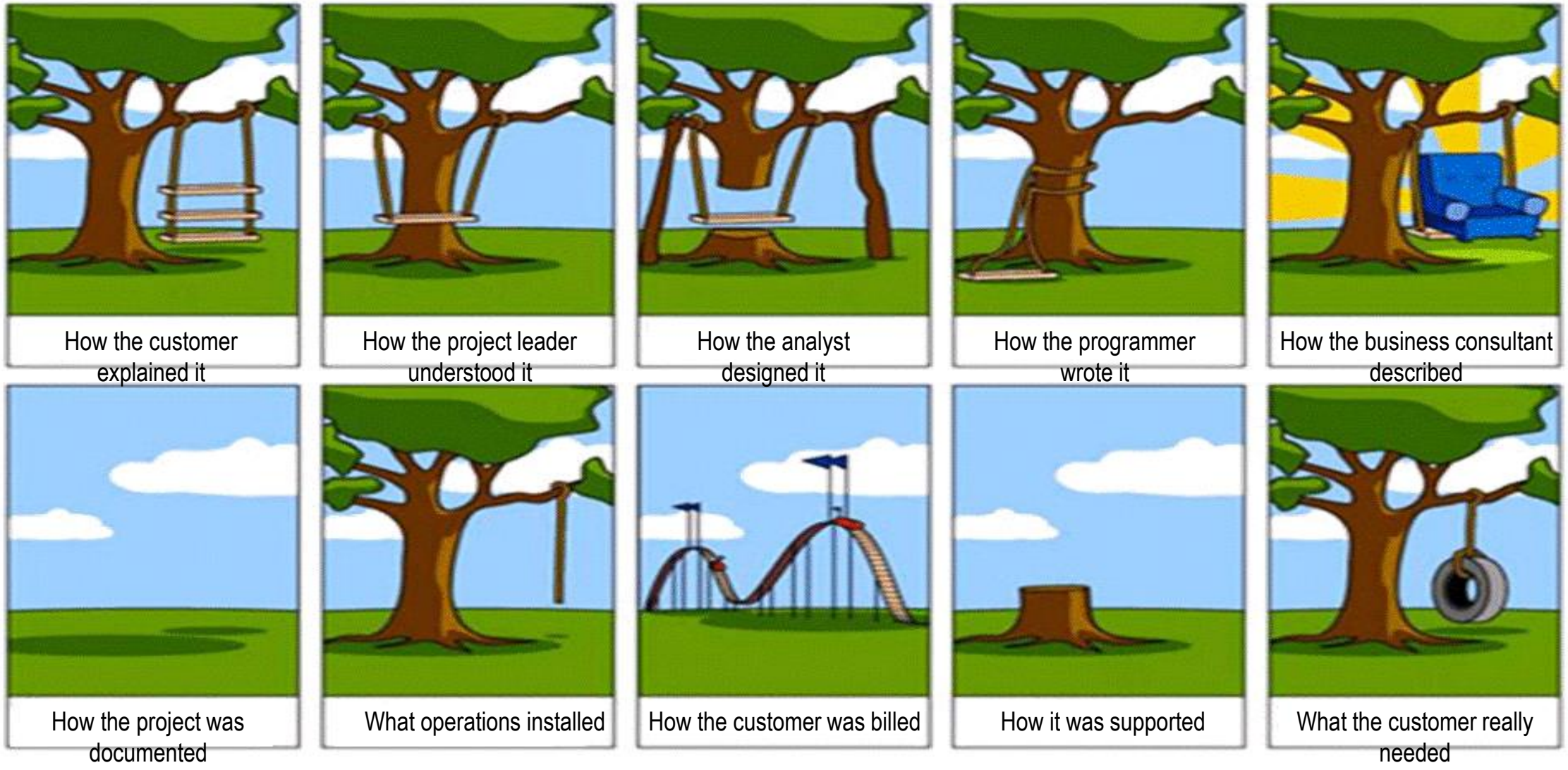
Contents

- Again from the IEEE standard
- The basic issues that the SRS writer(s) shall address are the following:
  - Functionality - What is the software supposed to do?
  - External interfaces - How does the software interact with people, the system's hardware, other hardware, and other software?
  - Performance - What is the speed, availability, response time, recovery time of various software functions, etc.?
  - Attributes - What are the portability, correctness, maintainability, security, etc. considerations?
  - Design constraints imposed on an implementation - Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

Characteristics

- Correct – what is asked by the user is correctly specified in it. Reviews ensure correctness.
- Unambiguous – it has only one interpretation. 'Software should be flexible' is an ambiguous requirement
- Complete – all aspects (functional, non-functional) of a requirement are specified with all combination of its flow (e.g. login procedure).
- Consistent – no two requirements conflict with each other.
- Ranked for importance and/or stability – indicates its relative importance.
- Verifiable – there should be some methods to verify that this requirement is implemented correctly. 'It should have good UI' is not a verifiable requirement.
- Modifiable – changes to the requirement can be made easily and consistently
- Traceable – one should be able to trace the requirement forward to its design component, source code, test case and reverse also.

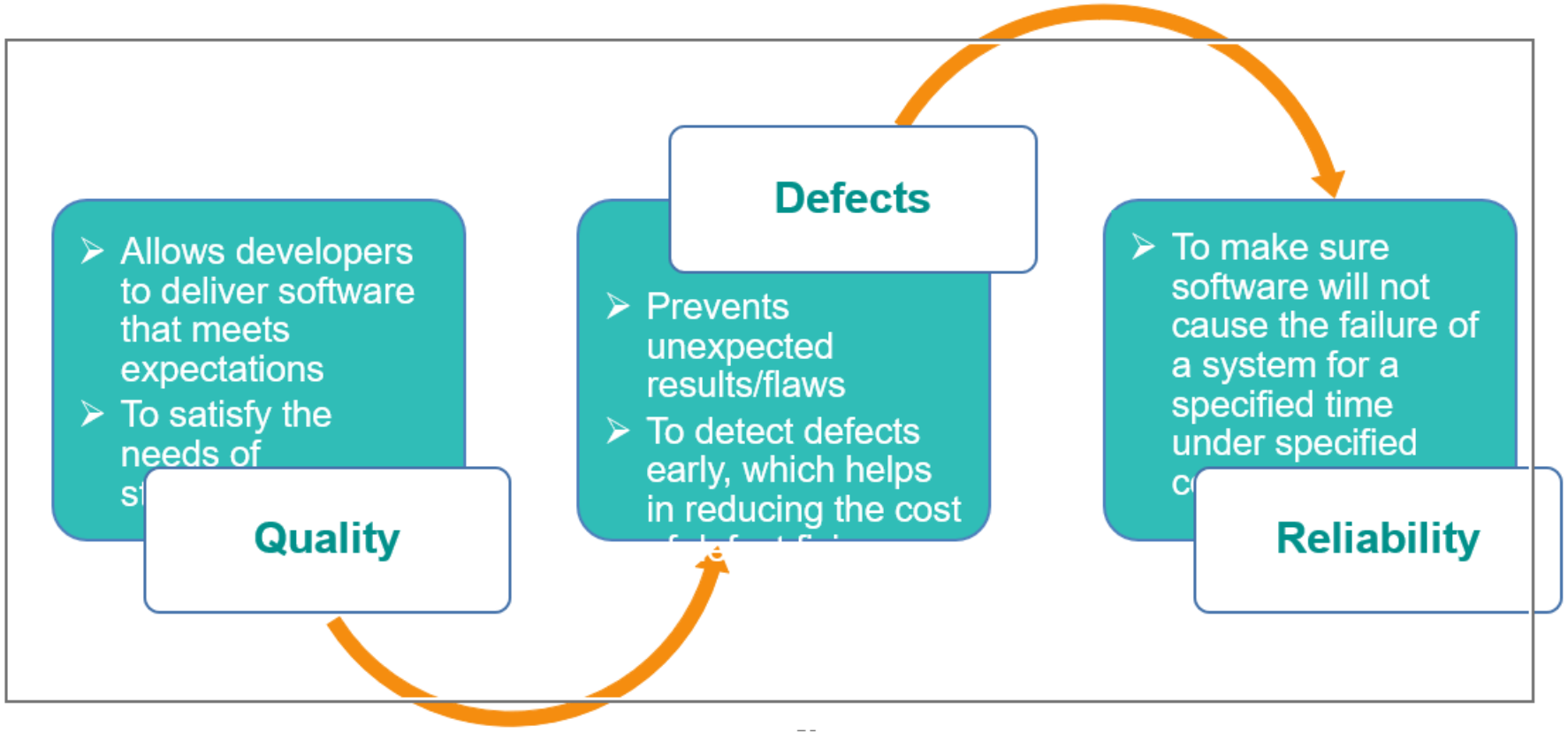
Importance of Good Requirements



Software Quality Assurance

- Process of evaluating the quality of product and procedures by enforcing adherence to software standards
- An umbrella activity that ensures conformance to standard and procedures throughout the SDLC of a software product
- **Quality** - A measurable characteristic or attribute of something
- **Quality Control** - Involves the series of inspections, reviews, and tests used throughout the software process
- **Quality Assurance** - Consists of the auditing and reporting functions of management

Importance of Software Quality Assurance



Software Quality Assurance Tasks

- Formulating a quality management plan
- Applying software engineering techniques
- Conducting formal technical reviews
- Applying a multi-tiered testing strategy
- Enforcing process adherence
- Controlling change
- Measuring the impact of change
- Performing QA audits
- Keeping records and reporting

Software Quality Assurance Benefits

- Improving maturity of the organization
- Ensuring effectiveness of the processes being followed
- Facilitating to tailor the procedures for the users
- Be the eyes and ears of top management
- Improving customers confidence
- Facilitating the team to make quality deliverables
- Controlling the cost of quality
- Getting more benefits out of certifications

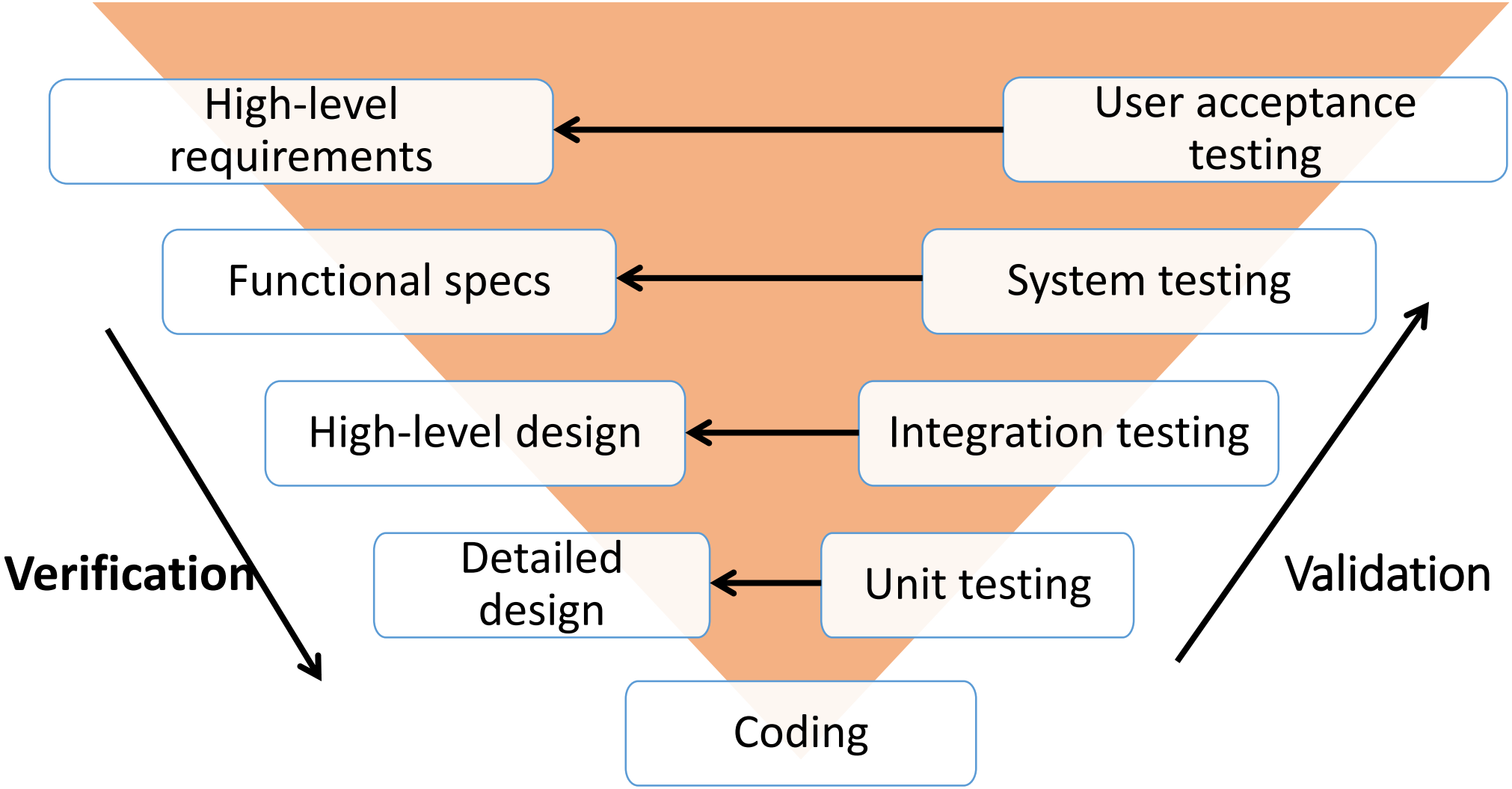
Software Testing

- A process of demonstrating that errors are not present
- A way of establishing confidence that a program does what it is supposed to do
- A means of achieving an error-free program by finding all errors
- A process of executing a program with the intent of finding errors
- A “DESTRUCTIVE”, yet creative process

Why Software Testing?

- Verifies that all requirements are implemented correctly (both for positive and negative conditions)
- Identifies defects before software deployment
- Helps improve quality and reliability
- Makes software predictable in behavior
- Reduces incompatibility and interoperability issues
- Helps marketability and retention of customers

V - Model


High-level requirements
User acceptance testing
Functional specs
System testing
High-level design
Integration testing
Detailed design
Unit testing
Coding
Verification
Validation